



CHARACTERIZING COMPONENT HIDING USING
ANCESTRAL ENTROPY

THESIS

Jason A. Williams, Captain, USAF

AFIT/GCE/ENG/09-12

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GCE/ENG/09-12

CHARACTERIZING COMPONENT HIDING USING ANCESTRAL
ENTROPY

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the
Degree of Master of Science in Computer Engineering

Jason A. Williams, BSCE

Captain, USAF

March 26, 2009

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.


CHARACTERIZING COMPONENT HIDING USING ANCESTRAL
ENTROPY

Jason A. Williams, BSCE
Captain, USAF

Approved:


LtCol Jeffrey T. McDonald, PhD (Chairman)

10 MAR 09
Date


Dr. Yong C. Kim (Member)

10 MAR 09
Date


Dr. Rusty O. Baldwin (Member)

10 Mar 09
Date

Abstract

In this research, the problem of software protection and the attributes that define that protection is considered. Specifically, how to protect programs defined as structural combinational logic gates. Obfuscation is one technique for protecting such circuits and involves replacing an original circuit with a functionally equivalent variant that has some definable hiding property. The difficulty of reverse engineering versus identifying and recovering the original components or sub-circuits within an original circuit is compared. With a polymorphic circuit engine that produces semantically equivalent variations of standard benchmark circuits the level of component hiding across variants with different physical configurations is determined to provide an entropy-based attribute to assess whether components are merged at the structural level. Specific types of obfuscating transformations with respect to component hiding using ancestral entropy are compared as well as the measure of uncertainty related to origination of a gate within a circuit.

Acknowledgements

First and foremost, I owe a large debt of gratitude to my wife for suggesting I attend AFIT, supporting me in preparing this thesis and throughout my military career. Any success of mine is equally yours. To my daughters, I thank you for reminding me what is truly important. I love you all very much.

Professionally, I owe a debt of gratitude to my thesis advisor, Lt Col Todd McDonald, Dr. Yong Kim, and Maj Eric Trias for asking the hard questions and providing guidance. Thank you.

Jason A. Williams

Table of Contents

| | Page |
|---|-------|
| Abstract | iv |
| Acknowledgements | v |
| Table of Contents | vi |
| List of Figures | ix |
| List of Tables | xv |
| List of Symbols | xvii |
| List of Abbreviations | xviii |
| I. Introduction | 1 |
| 1.1 Scope | 1 |
| 1.2 Problem Definition | 2 |
| 1.2.1 Airframes | 2 |
| 1.2.2 Spacecraft | 3 |
| 1.2.3 Hainan Island | 3 |
| 1.3 Reverse Engineering Legality | 5 |
| 1.4 Goals and Hypothesis | 5 |
| 1.5 Organization | 6 |
| II. Literature Review | 7 |
| 2.1 Obfuscation | 7 |
| 2.2 Black Box and White Box Analysis | 7 |
| 2.3 Reverse Engineering | 8 |
| 2.4 Embedded Systems | 11 |
| 2.5 Side channel attacks | 12 |
| 2.6 Invasive attacks | 13 |
| 2.7 Algorithm Overview | 13 |
| 2.7.1 Gate Selection | 14 |
| 2.7.2 Gate Replacement | 15 |
| 2.8 Selection and Replacement Example | 18 |
| 2.9 Gate Level Ancestry | 22 |
| 2.9.1 Ancestry File Structure | 23 |
| 2.9.2 Computing Ancestry | 24 |
| 2.10 Entropy | 27 |

| | Page |
|---|------|
| III. Methodology | 29 |
| 3.1 Problem Definition | 29 |
| 3.1.1 Goals and Hypothesis | 29 |
| 3.1.2 Approach | 30 |
| 3.2 System boundaries | 30 |
| 3.3 System Services | 30 |
| 3.4 Workload | 31 |
| 3.5 System Parameters | 32 |
| 3.6 Performance Metrics | 33 |
| 3.7 Factors | 34 |
| 3.8 Evaluation Technique | 34 |
| 3.9 Experimental Design | 37 |
| 3.10 Adding Color and Shape to each Gate | 37 |
| 3.11 Component Hiding | 42 |
| 3.12 Methodology Summary | 43 |
| IV. Results | 45 |
| 4.1 Three Gate Replacement C-17 Series Circuits | 45 |
| 4.1.1 FixedLevelTwoGates Experiment | 46 |
| 4.1.2 OutputLevelTwoGates | 50 |
| 4.1.3 LargestLevelTwoGates | 51 |
| 4.1.4 RandomLevelTwoGates | 52 |
| 4.1.5 RandomTwoGates | 53 |
| 4.1.6 RandomAlgorithm | 55 |
| 4.1.7 Series Circuits Results Summary | 56 |
| 4.2 Three Gate Replacement C-17 Parallel Circuits | 64 |
| 4.2.1 FixedLevelRandomTwoGates | 65 |
| 4.2.2 OutputLevelRandomTwoGates | 66 |
| 4.2.3 LargestLevelRandomTwoGates | 66 |
| 4.2.4 RandomLevelTwoGates | 67 |
| 4.2.5 RandomTwoGates | 68 |
| 4.2.6 RandomAlgorithm | 69 |
| 4.2.7 Parallel Circuits Results Summary | 70 |
| 4.3 Three and Four Gate Replacement Circuit Reduction | 71 |
| 4.4 Four Gate Replacement C-17 Series Circuits | 81 |
| 4.5 Four Gate Replacement C-17 Parallel Circuits | 81 |
| 4.6 Validation | 97 |
| 4.7 Summary | 98 |

| | Page |
|--|------|
| V. Conclusions | 101 |
| 5.1 Goals and Hypothesis | 101 |
| 5.2 Contributions | 101 |
| 5.2.1 Component Hiding Effectiveness | 101 |
| 5.2.2 Introduction of the Colored Graphs | 101 |
| 5.3 Future Work | 102 |
| 5.3.1 Color Graph Improvements | 102 |
| 5.3.2 Execute at Runtime | 102 |
| 5.3.3 Modify FixedLevelTwoGates Algorithm | 102 |
| 5.3.4 Validation | 103 |
| Appendix A. Three Gate Replacement Series Circuit Variant Graphs and Charts | 104 |
| Appendix B. Three Gate Replacement Parallel Circuit Variant Graphs and Charts | 140 |
| Appendix C. Four Gate Replacement Series and Parallel Circuit Variant Charts and Graphs | 164 |
| Bibliography | 179 |
| Vita | 181 |

List of Figures

| Figure | | Page |
|--------|--|------|
| 1.1 | B-29 and Tu-4 | 3 |
| 1.2 | B-1 Lancer and Tu-160 BlackJack | 3 |
| 1.3 | U.S. Shuttle and Russian Buran | 4 |
| 1.4 | Spacecraft in Transport | 4 |
| 2.1 | C432 High Level Abstraction | 9 |
| 2.2 | 74238 Carry Look-Ahead Adder | 10 |
| 2.3 | Directed and Undirected Graphs | 15 |
| 2.4 | Cycle in a Replacement Circuit | 17 |
| 2.5 | Two Gate Selection | 18 |
| 2.6 | Three Gate Replacement with New Control Flow | 19 |
| 2.7 | C-17 Benchmark Circuit and BENCH File | 20 |
| 2.8 | Graph Representing C-17 Benchmark Circuit | 20 |
| 2.9 | Selection and Replacement Example | 21 |
| 2.10 | Iteration 2 Selection and Replacement | 21 |
| 2.11 | Iteration 3 Selection and Replacement | 22 |
| 2.12 | Baseline Circuit with Ancestry | 24 |
| 2.13 | 1 st Iteration | 25 |
| 2.14 | Ancestry Example Circuit, 15 th Iteration | 26 |
| 3.1 | Circuit Transformation System (CORGI) | 31 |
| 3.2 | C-17 Series Circuits | 34 |
| 3.3 | High Level C-17 Parallel Circuits | 35 |
| 3.4 | File Creation Flow Chart | 36 |
| 3.5 | C-17 Parallel Circuit Maintaining Structure | 38 |
| 3.6 | C-17 Series Circuit Maintaining Structure | 38 |
| 3.7 | Basic Graph of a Circuit | 39 |
| 3.8 | Matching Results for 3 Levels | 41 |

| Figure | | Page |
|--------|--|------|
| 3.9 | Component Hiding Cases | 44 |
| 4.1 | High Level Series C-17 Circuits | 46 |
| 4.2 | Fixed Level C-17 5 Input Circuits | 48 |
| 4.3 | Average Ancestral Entropy Intervals for 5 Input Series Circuit . . . | 57 |
| 4.4 | Average Ancestral Entropy Intervals for 5 ‘Split’ Input Series Circuit | 58 |
| 4.5 | Average Ancestral Entropy Intervals for 11 Input Series Circuit . . | 59 |
| 4.6 | Percentage of Original Gates in Series Circuits | 60 |
| 4.7 | Boxplot from 2-Sample t Test of Average Output Ancestral Entropy with Random Algorithms on Series Circuits | 61 |
| 4.8 | Boxplot from 2-Sample t Test of Average Node Ancestral Entropy with Random Algorithms on Series Circuits | 62 |
| 4.9 | Boxplot from 2-Sample t test of Average Level Ancestral Entropy with Random Algorithms on Series Circuits | 63 |
| 4.10 | C-17 Parallel Circuits | 64 |
| 4.11 | Average Ancestral Entropy Intervals for Shared Input Parallel Circuit | 72 |
| 4.12 | Average Ancestral Entropy Intervals for Individual Input Parallel Circuit | 73 |
| 4.13 | Percentage of Original Gates in Parallel Circuits | 74 |
| 4.14 | Boxplot from 2-Sample t Test of Average Output Ancestral Entropy with Random Algorithms on Parallel Circuits | 75 |
| 4.15 | Boxplot from 2-Sample t Test of Average Output Ancestral Entropy with Random Algorithms on Parallel Circuits | 76 |
| 4.16 | Boxplot from 2-Sample t Test for Average Output Ancestral Entropy with Random Algorithms on Parallel Circuits | 77 |
| 4.17 | 3 Gate Replacement C-17 Circuit Variant at 1000 iterations | 78 |
| 4.18 | 3 Gate Replacement C-17 Circuit Reduced | 78 |
| 4.19 | 4 Gate Replacement C-17 Circuit Variant at 1000 iterations | 79 |
| 4.20 | 4 Gate Replacement C-17 Circuit Reduced | 80 |
| 4.21 | Average Ancestral Entropy Intervals for 5 Input Series Circuit with a 4 Gate Replacement | 83 |

| Figure | | Page |
|--------|---|------|
| 4.22 | Average Ancestral Entropy Intervals for 5 ‘Split’ Input Series Circuit with a 4 Gate Replacement | 84 |
| 4.23 | Average Ancestral Entropy Intervals for 11 Input Series Circuit with a 4 Gate Replacement | 85 |
| 4.24 | Percentage of Original Gates in Series Circuits | 86 |
| 4.25 | Boxplot from 2-Sample t Test for Average Output Ancestral Entropy with Random Algorithms on Series Circuits with a 4 Gate Replacement | 87 |
| 4.26 | Boxplot from 2-Sample t Test for Average Node Ancestral Entropy with Random Algorithms on Series Circuits with a 4 Gate Replacement | 88 |
| 4.27 | Boxplot from 2-Sample t Test for Average Level Ancestral Entropy with Random Algorithms on Series Circuits with a 4 Gate Replacement | 89 |
| 4.28 | Average Ancestral Entropy Intervals for Shared Input Parallel Circuit with a 4 Gate Replacement | 92 |
| 4.29 | Average Ancestral Entropy Intervals for Individual Input Parallel Circuit with a 4 Gate Replacement | 93 |
| 4.30 | Boxplot from 2-Sample t Test of Average Output Ancestral Entropy with Random Algorithms on Parallel Circuits | 94 |
| 4.31 | Boxplot from 2-Sample t Test of Average Node Ancestral Entropy with Random Algorithms on Parallel Circuits | 95 |
| 4.32 | Boxplot from 2-Sample t Test of Average Level Ancestral Entropy with Random Algorithms on Parallel Circuits | 96 |
| 4.33 | Baseline Individual Input Parallel C-17 Circuit | 98 |
| 4.34 | Reducing C-17 Parallel Circuit with Duplicate Inputs and Redundant Gates to Independent Components | 99 |
| 4.35 | Reducing C-17 Parallel Circuit with Duplicate Inputs and Redundant Gates One Iteration Before Component Separation | 99 |
| 4.36 | Reducing C-17 Parallel Circuit with No Duplicate Inputs or Redundant Gates to Independent Components | 100 |
| 4.37 | Reducing C-17 Parallel Circuit with No Duplicate Inputs or Redundant Gates One Iteration Before Component Separation | 100 |
| A.1 | FixedLevelTwoGates 5 Input C-17 Series Variant Circuits | 104 |

| Figure | | Page |
|--------|---|------|
| A.2 | FixedLevelTwoGates 5 Input C-17 Series Graphs | 105 |
| A.3 | FixedLevelTwoGates 5 ‘Split’ Input C-17 Series Variant Circuits . | 106 |
| A.4 | FixedLevelTwoGates 5 ‘Split’ Input C-17 Series Graphs | 107 |
| A.5 | FixedLevelTwoGates 11 Input C-17 Series Variant Circuits | 108 |
| A.6 | FixedLevelTwoGates 11 Input C-17 Series Graphs | 109 |
| A.7 | OutputLevelTwoGates 5 Input C-17 Series Variant Circuits | 110 |
| A.8 | OutputLevelTwoGates 5 Input C-17 Series Graphs | 111 |
| A.9 | OutputLevelTwoGates 5 ‘Split’ Input C-17 Series Variant Circuits | 112 |
| A.10 | OutputLevelTwoGates 5 ‘Split’ Input C-17 Series Graphs | 113 |
| A.11 | OutputLevelTwoGates 11 Input C-17 Series Variant Circuits . . . | 114 |
| A.12 | OutputLevelTwoGates 11 Input C-17 Series Graphs | 115 |
| A.13 | LargestLevelTwoGates 5 Input C-17 Series Variant Circuits | 116 |
| A.14 | LargestLevelTwoGatesl 5 Input C-17 Series Graphs | 117 |
| A.15 | LargestLevelTwoGates 5 ‘Split’ Input C-17 Series Variant Circuits | 118 |
| A.16 | LargestLevelTwoGates 5 ‘Split’ Input C-17 Series Graphs | 119 |
| A.17 | LargestLevelTwoGates 11 Input C-17 Series Variant Circuits . . . | 120 |
| A.18 | LargestLevelTwoGates 11 Input C-17 Series Graphs | 121 |
| A.19 | RandomLevelTwoGates 5 Input C-17 Series Variant Circuits . . . | 122 |
| A.20 | RandomLevelTwoGates 5 Input C-17 Series Graphs | 123 |
| A.21 | RandomLevelTwoGates 5 ‘Split’ Input C-17 Series Variant Circuits | 124 |
| A.22 | RandomLevelTwoGates 5 ‘Split’ Input C-17 Series Graphs | 125 |
| A.23 | RandomLevelTwoGates 11 Input C-17 Series Variant Circuits . . | 126 |
| A.24 | RandomLevelTwoGates 11 Input C-17 Series Graphs | 127 |
| A.25 | RandomTwoGates 5 Input C-17 Series Variant Circuits | 128 |
| A.26 | RandomTwoGates 5 Input C-17 Series Graphs | 129 |
| A.27 | RandomTwoGates 5 ‘Split’ Input C-17 Series Variant Circuits . . | 130 |
| A.28 | RandomTwoGates 5 ‘Split’ Input C-17 Series Graphs | 131 |
| A.29 | RandomTwoGates 11 Input C-17 Series Variant Circuits | 132 |

| Figure | | Page |
|--------|--|------|
| A.30 | RandomTwoGates 11 Input C-17 Series Graphs | 133 |
| A.31 | RandomAlgorithm 5 Input C-17 Series Variant Circuits | 134 |
| A.32 | RandomAlgorithm 5 Input C-17 Series Graphs | 135 |
| A.33 | RandomAlgorithm 5 ‘Split’ Input C-17 Series Variant Circuits . . | 136 |
| A.34 | RandomAlgorithm 5 ‘Split’ Input C-17 Series Graphs | 137 |
| A.35 | RandomAlgorithm 11 Input C-17 Series Variant Circuits | 138 |
| A.36 | RandomAlgorithm 11 Input C-17 Series Graphs | 139 |
| B.1 | FixedLevelTwoGates Shared Input C-17 Parallel Variant Circuits | 140 |
| B.2 | FixedLevelTwoGates Shared Input C-17 Parallel Charts | 141 |
| B.3 | FixedLevelTwoGates Individual Input C-17 Parallel Variant Cir- cuits | 142 |
| B.4 | FixedLevelTwoGates Individual Input C-17 Parallel Charts | 143 |
| B.5 | OutputLevelTwoGates Shared Input C-17 Parallel Variant Circuits | 144 |
| B.6 | OutputLevelTwoGates Shared Input C-17 Parallel Charts | 145 |
| B.7 | OutputLevelTwoGates Individual Input C-17 Parallel Variant Cir- cuits | 146 |
| B.8 | OutputLevelTwoGates Individual Input C-17 Parallel Charts . . . | 147 |
| B.9 | LargestLevelTwoGates Shared Input C-17 Parallel Variant Circuits | 148 |
| B.10 | LargestLevelTwoGates Shared Input C-17 Parallel Charts | 149 |
| B.11 | LargestLevelTwoGates Individual C-17 Parallel Variant Circuits . | 150 |
| B.12 | LargestLevelTwoGates Individual Input C-17 Parallel Charts . . . | 151 |
| B.13 | RandomLevelTwoGates Shared Input C-17 Parallel Variant Circuits | 152 |
| B.14 | RandomLevelTwoGates Shared Input C-17 Parallel Charts | 153 |
| B.15 | RandomLevelTwoGates Individual Input C-17 Parallel Variant Cir- cuits | 154 |
| B.16 | RandomLevelTwoGates Individual Input C-17 Parallel Charts . . . | 155 |
| B.17 | RandomTwoGates Shared Input C-17 Parallel Variant Circuits . . | 156 |
| B.18 | RandomTwoGates Shared Input C-17 Parallel Charts | 157 |
| B.19 | RandomTwoGates Individual Input C-17 Parallel Variant Circuits | 158 |

| Figure | | Page |
|--------|---|------|
| B.20 | RandomTwoGates Individual Input C-17 Parallel Charts | 159 |
| B.21 | RandomAlgorithm Shared Input C-17 Parallel Variant Circuits . . | 160 |
| B.22 | RandomAlgorithm Shared Input C-17 Parallel Charts | 161 |
| B.23 | RandomAlgorithm Individual Input C-17 Parallel Variant Circuits | 162 |
| B.24 | RandomAlgorithm Individual Input C-17 Parallel Charts | 163 |
| C.1 | RandomLevelTwoGates 5 Input C-17 Series 4 Gate Replacement Charts | 164 |
| C.2 | RandomLevelTwoGates 5 ‘Split’ Input C-17 Series 4 Gate Replace- ment Charts | 165 |
| C.3 | RandomLevelTwoGates 11 Input C-17 Series 4 Gate Replacement Charts | 166 |
| C.4 | RandomTwoGates 5 Input C-17 Series 4 Gate Replacement Charts | 167 |
| C.5 | RandomTwoGates 5 ‘Split’ Input C-17 Series 4 Gate Replacement Charts | 168 |
| C.6 | RandomTwoGates 11 Input C-17 Series 4 Gate Replacement Charts | 169 |
| C.7 | RandomAlgorithm 5 Input C-17 Series 4 Gate Replacement Charts | 170 |
| C.8 | RandomAlgorithm 5 ‘Split’ Input C-17 Series 4 Gate Replacement Charts | 171 |
| C.9 | RandomAlgorithm 11 Input C-17 Series 4 Gate Replacement Charts | 172 |
| C.10 | 4 Gate Replacement RandomLevelTwoGates Individual Input C-17 Parallel Charts | 173 |
| C.11 | 4 Gate Replacement RandomTwoGates Individual Input C-17 Par- allel Charts | 174 |
| C.12 | 4 Gate Replacement RandomAlgorithm Individual Input C-17 Par- allel Charts | 175 |
| C.13 | 4 Gate Replacement RandomLevelTwoGates Shared Input C-17 Par- allel Charts | 176 |
| C.14 | 4 Gate Replacement RandomTwoGates Shared Input C-17 Parallel Charts | 177 |
| C.15 | 4 Gate Replacement RandomAlgorithm Shared Input C-17 Parallel Charts | 178 |

List of Tables

| Table | | Page |
|-------|--|------|
| 2.1 | ISCAS '85 Benchmark Circuits | 8 |
| 2.2 | Results per Iteration for RandomTwoGate Algorithm | 19 |
| 2.3 | Experiment BENCH Files | 20 |
| 2.4 | BENCH and Ancestry File Example | 23 |
| 2.5 | Maximum Ancestral Entropy Values | 28 |
| 3.1 | Factors and Levels | 35 |
| 3.2 | Gate Operation and Matching Shape | 40 |
| 3.3 | Coloring Schemes | 40 |
| 4.1 | C-17 Series Experiments Ancestry Files | 47 |
| 4.2 | P-Values from 2-Sample t Test comparing RandomAlgorithm against RandomTwoGates and RandomLevelTwoGates Using Series Circuits | 64 |
| 4.3 | P-Values from 2-Sample t Test comparing RandomAlgorithm against RandomTwoGates and RandomLevelTwoGates Using Parallel Cir- cuits | 71 |
| 4.4 | RandomLevelTwoGates 3 and 4 Gate Replacement Comparison, Se- ries Circuits | 82 |
| 4.5 | RandomTwoGates 3 and 4 Gate Replacement Comparison, Series Circuits | 82 |
| 4.6 | RandomAlgorithm 3 and 4 Gate Replacement Comparison, Series Circuits | 82 |
| 4.7 | P-Values from 2-Sample t Test comparing RandomAlgorithm against RandomTwoGates and RandomLevelTwoGates Using Series Cir- cuits with a 4 Gate Replacement | 82 |
| 4.8 | RandomLevelTwoGates 3 and 4 Gate Replacement Comparison, Parallel Circuits | 90 |
| 4.9 | RandomTwoGates 3 and 4 Gate Replacement Comparison, Parallel Circuits | 90 |
| 4.10 | RandomAlgorithm 3 and 4 Gate Replacement Comparison, Parallel Circuits | 90 |

| Table | | Page |
|-------|--|------|
| 4.11 | P-Values from 2-Sample t Test comparing RandomAlgorithm against RandomTwoGates and RandomLevelTwoGates Using Parallel Circuits with a 4 Gate Replacement | 91 |
| 4.12 | Reduction Results | 97 |

List of Symbols

| Symbol | | Page |
|----------|---|------|
| C | Original, un-obfuscated circuit | 13 |
| Ω | Gate replacement basis | 13 |
| Φ | The set of all gates in a circuit | 14 |
| γ | Gate Selection Strategy | 15 |
| S | The size of a circuit | 16 |

List of Abbreviations

| Abbreviation | | Page |
|--------------|---|------|
| DoD | Department of Defense | 1 |
| AT | Anti-Tamper | 1 |
| EPITS | Essential Program Information Technologies | 1 |
| RDT&E | Research, Development, Test and Evaluation | 2 |
| SCPA | Semiconductor Chip Protection Act | 5 |
| ISCAS | International Symposium on Circuits and Systems | 7 |
| SEC | Single Error Correcting | 8 |
| ALU | Arithmetic Logic Unit | 8 |
| DED | Double Error Detecting | 8 |
| CLA | Carry Look Ahead | 10 |
| RFID | Radio Frequency Identification | 11 |
| EMA | Electromagnetic analysis | 12 |
| CORGI | Circuit Obfuscation via Randomization of Graphs Iteratively . | 13 |
| DAG | Directed Acyclic graph | 15 |
| SUT | System Under Test | 30 |
| RBG | Red, Blue, and Green | 102 |

CHARACTERIZING COMPONENT HIDING USING ANCESTRAL ENTROPY

I. Introduction

Current military operations have United States military assets and technologies spread throughout the world. Many of these operations include multinational forces with access to U.S. equipment. Agreements with our allies and the Department of Defense (DoD) contractors to participate in foreign sales, the loss or capture of military assets, and overseas computer product fabrication increase the risk of our adversaries reverse engineering U.S. equipment. To reduce access to information, the DoD has established an anti-tamper (AT) policy to protect U.S. assets. DOD 5200.1-M defines the processes by which information, technologies and systems that are essential to the successful development of new DoD systems are identified and protected. Essential Program Information Technologies, and/or Systems (EPITS) are the critical elements of a system that make it unique and valuable to the U.S. defense forces. The EPITS are those items that would cause degradation of combat effectiveness, shorten the lifetime, or allow a foreign activity to neutralize a U.S. system [5]. To protect this critical information, the U.S. military uses a wide variety of AT techniques.

1.1 Scope

This thesis uses a circuit variant creator to mask the physical appearance of circuit components to complicate particular reverse engineering efforts. Entropy serves to identify if a gate from a circuit or sub-circuit variant is derived from a specific component of the original circuit.

1.2 Problem Definition

Software and hardware obfuscation is essential when designing critical equipment that an adversary may obtain. The DoD budgeted over 75 billion dollars for Research, Development, Test and Evaluation (RDT&E) in 2008 [4]. When an adversary doesn't have to spend the time or money to develop equivalent equipment, the financial investment in developing the equipment and any time is shortened. If an adversary has complete or partial access to DoD equipment or designs they gain information the DoD didn't intend them to have. Just a few examples are both new and old technologies, U.S. military and commercial design strategies, and information gathering techniques. Below, several examples illustrate this very thing.

1.2.1 Airframes. During WWII, Joseph Stalin refused to allow American B-29 aircraft to land in Russia because they were not at war with Japan and did not want to fight a war on two fronts. Unfortunately, several U.S. crews were forced to divert to Russia and land after completing bombing missions on Japan and the aircraft and crew were detained by the Russians. The crew were eventually returned, but three B-29 Superfortresses, two damaged and one completely in tact, were kept by Russia and never returned. Stalin thought after the war that he would need long range bomber capability to reach North America to prevent any aggression from the Americans. Estimating it would take five years to develop his own long range aircraft, he copied the B-29 bolt for bolt [11], saving development time and money. The B-39 Superfortress is shown in the top half of Figure 1.1; the bottom half is the Russian Tu-4.

One Russian military aircraft that bears a striking resemblance to U.S. aircraft is the Tu-160 long range bomber. As Figure 1.2 shows, both aircraft have the same basic shape, size, and both have variable-sweep wings. Although many aircraft are designed to counter a threat from an adversary, it is most likely no coincidence that the resulting aircraft appear so similar.

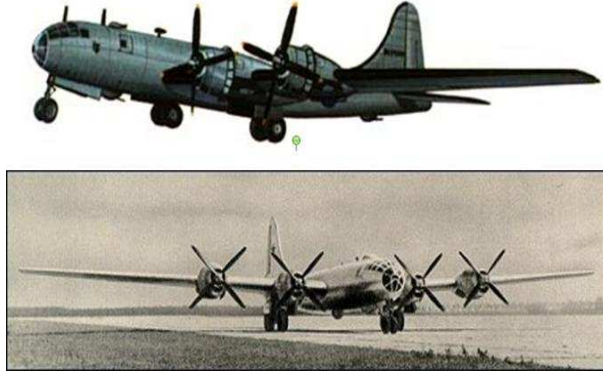


Figure 1.1: B-29 and Tu-4



Figure 1.2: B-1 Lancer and Tu-160 BlackJack

1.2.2 Spacecraft. Another example of U.S. developed equipment copied by our adversaries is the NASA Space Shuttle. The Russian Buran orbiter is strikingly similar to the U.S. version, as shown in Figure 1.3. The Russians copied as much from the American version of the space shuttle as possible to guarantee the successful development of the Buran orbiter [20]. Not surprisingly, the Russians also copied the U.S. idea for transporting the orbiter. As Figure 1.4 illustrates, the Russians used the AN-225 MYIRA aircraft to transport their orbiter where NASA uses a modified Boeing 747.

1.2.3 Hainan Island. In 2001 a U.S. EP-3 reconnaissance aircraft and a Chinese Shenyang J-8 collided in international airspace approximately 70 miles from the Chinese island of Hainan. Due to the damage from the accident, the EP-3 was

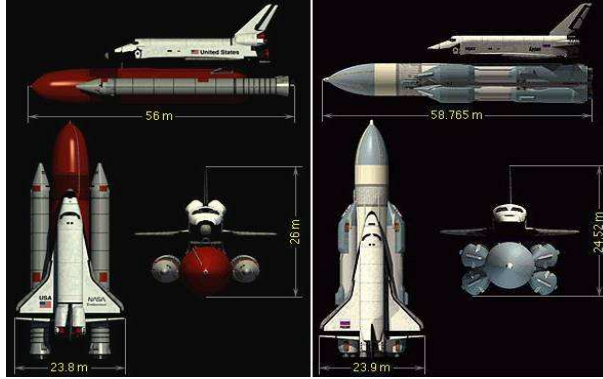


Figure 1.3: U.S. Shuttle and Russian Buran



Figure 1.4: Spacecraft in Transport

forced to make an emergency landing on Hainan Island. The crew was released after 12 days, but the aircraft was in the hands of the Chinese for a total of 94 days. According to a UPI press release [10], the EP-3 couldn't have landed at a better place for China and a worse place for U.S. military intelligence,

“Hainan island is host to one of China’s largest electronic-signals-intelligence complexes and is manned by experts who can glean critical information on the aircraft’s capabilities if they gain access to the Navy’s EP-3...”
Pentagon sources said.

It is unknown what information the Chinese learned from having access to this reconnaissance aircraft, but this is a prime example of why anti-tamper measures are essential. Effective hardware and software protection need to be developed to protect U.S. technologies.

1.3 *Reverse Engineering Legality*

The Semiconductor Chip Protection Act of 1984 (SCPA) stipulates that copying all or part of a chip for purposes of analyzing its layout is legal. However, simply copying a design is illegal. For copying to be legal, the reverse engineered chip must be substantially different than the original. For the chip manufacturer to protect the product, it is in their best interest to design it with a reverse engineer in mind, keeping the layout as difficult to understand as possible [18]. Although the law doesn't apply internationally and will not be respected by our military adversaries, it is important to follow this guidance to protect U.S. financial investments and military technologies. The SCPA sets the stage for research in circuit level obfuscation and circuit component hiding to increase the difficulty of reverse engineering U.S. technology.

1.4 *Goals and Hypothesis*

If an adversary attempts to reverse engineer a circuit variant, how can the difficulty of this effort in comparison to the reverse engineering the original circuit be characterized? If a new gate is introduced into a circuit derived from two independent circuits or sub-circuits, what is the level of difficulty for the reverse engineer to undo this transformation? An analysis of circuit transformations is performed on functionally equivalent circuits to define a measure of hiding specific circuit components. The first goal of this research is to define the term *component hiding*. Commonly used and easily identifiable components or sub-components of a circuit are identified and their structure changed to increase the difficulty of reverse engineering. Hiding these components is critical to thwarting some reverse engineering techniques. The second goal of this thesis is to compare circuit transformations that create circuit variants to identify component hiding properties. The third goal is to define an attribute that reflects the obfuscating properties a circuit variant provides. This protects programs from being easily copied and protects the DoD's financial investment as well as increasing the level of difficulty for our adversaries to reverse engineer U.S. military technology. Can this be accomplished using a polymorphic circuit engine based on

iterative selection and replacement? Can the uncertainty of component identification and the work required by an adversarial reverse engineer be measured? To test this different circuit configurations are used, how sub-circuits are merged by different selection and replacement algorithms and de-merged via a circuit reduction program is investigated.

1.5 Organization

This thesis is organized as follows. Chapter II provides the background research and sets the stage for this research effort. Chapter III describes in detail the method used to perform the experiments. Chapter IV describes the results of the experiments. Chapter V summarizes the contributions of this thesis and discusses future work in this research area. Appendix A contains figures illustrating the results of three gate replacement experiments performed using series circuits. Appendix B contains figures illustrating the results of three gate replacement experiments performed using parallel circuits. Appendix C contains figures illustrating the results of experiments performed using four gate replacement algorithms on series and parallel circuits.

II. Literature Review

2.1 *Obfuscation*

Obfuscation, according to Webster [13] means to “make obscure” or to “confuse”. Obfuscation in general means to make something harder to understand. In the computer world, obfuscation means to alter software by concealing the structure and intent of the code, while preserving the behavior [15]. Obfuscation can be viewed as a semantic-preserving transformation of a computer program that transforms the original program into a variant which complicates the understanding of the algorithm and data structures. It also prevents extracting valuable information from the source code of the program [16].

2.2 *Black Box and White Box Analysis*

The first type of analysis a reverse engineer could do is black box analysis. This is often looked at as analyzing the *behavior* of a circuit. McDonald considers software behavior as the black box functional characteristics of a circuit reflected by all possible input and output combinations, or denotational semantics [12]. In the case of black box analysis, the reverse engineer has no access to the internal workings of a circuit and doesn’t know the physical input and output relationship. He must try all possible combinations of inputs and measure each corresponding output to determine the function of the circuit. Limiting the reverse engineer to black box analysis is the best result a circuit obfuscator can achieve and this analysis is unavoidable for low level control logic circuits that are truly random [8]. Forcing the reverse engineer to try all combinations is effective even in relatively small circuits. For example, the International Symposium on Circuits and Systems (ISCAS) ’85 circuits listed in Table 2.1 have inputs that number from 32 to 233. The smallest circuit has 2^{32} , or 4,294,967,296 possible input combinations. At 1 second to test each input combination and record every output, all possibilities are covered in just over 136 years. If every test takes only 1 millisecond, it will still take nearly 50 days to test all combinations in this small circuit.

| Circuit | Function | Inputs | Outputs | Logic Gates | Major Functional Blocks |
|----------------|---------------------------------|---------------|----------------|--------------------|--------------------------------|
| C-432 | 27-channel interrupt controller | 36 | 7 | 160 | 5 |
| C-499 | 32-bit SEC circuit | 41 | 32 | 202 | 2 |
| C-880 | 8-bit ALU | 60 | 26 | 383 | 7 |
| C-1355 | 32-bit SEC circuit | 41 | 32 | 546 | 2 |
| C-1908 | 16-bit SEC/DED circuit | 33 | 25 | 880 | 6 |
| C-2670 | 12-bit ALU and controller | 233 | 140 | 1,193 | 7 |
| C-3540 | 8-bit ALU | 50 | 22 | 1,669 | 11 |
| C-5315 | 9-bit ALU | 178 | 123 | 2,307 | 10 |
| C-6288 | 16x16 multiplier | 32 | 32 | 2,406 | 240 |
| C-7552 | 32-bit adder and comparator | 207 | 108 | 3,512 | 8 |

Table 2.1: ISCAS '85 Benchmark Circuits

The second type of analysis is white box analysis. This analyzes circuit reconstruction, commonly referred to as the structural topology. In white box analysis, the reverse engineer has access to the sequence of the inputs and outputs and the internal gate structure of the circuit. To the reverse engineer, having white box access is a huge improvement because white box analysis might identify commonly used component structures, such as adders, multipliers, multiplexers, and accumulators. These common components can be grouped together to form a high level functional unit that is easier to understand. Figure 2.1 is a high level look at the 36 input, 160 gate C-432 benchmark circuit. If this can be converted from a series of gates into a high level module, time consuming process of black box analysis can be avoided.

2.3 Reverse Engineering

Reverse engineering, according Chikofsky and Cross [3], is

“...regularly applied to improve your own products, as well as analyze a competitor’s products or those of an adversary in a military or national-security situation.”

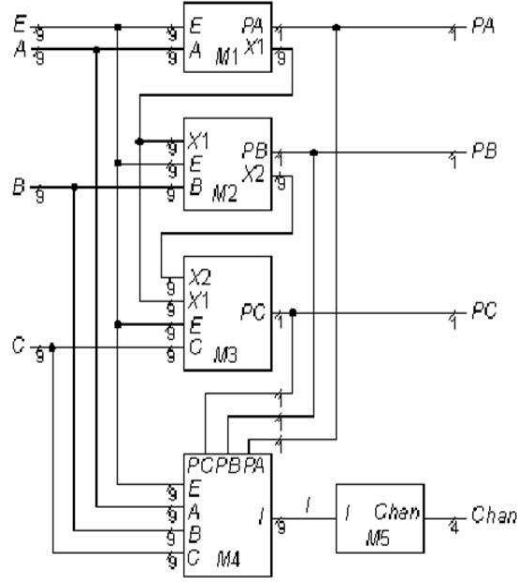


Figure 2.1: C432 High Level Abstraction

They continue to say that reverse engineering is the process of analyzing a system to identify the components and their relationships, and to create a representation of the system in another form or at a higher level of abstraction. It is a process of examination, not a process of change or replication. M.G Reckoff defines reverse engineering as

“the process of developing a set of specifications for a complex hardware system by an orderly examination of specimens of that system.”

The process is done by someone other than the developer of the system and without the benefit of having access to any of the original system drawings for the purpose of making a clone of the original hardware system [19].

Reverse engineers use many techniques to determine the functionality of a circuit component. One white-box technique described in [8], visibly looks for functional modules that are often used in industry to determine what the component or portion of a component does. Identifying these modules greatly simplifies reverse engineering by reducing the level of black box analysis. Even if the high-level designs are not available, looking for typical designs used in circuit production, identifying commonly

used sub-components of a design, or extracting the circuit from the net list makes it easier to transform the circuit into a high-level functional model. By hiding the visible signatures of these modules, a common reverse engineering technique can be eliminated. For example, in Figure 2.2 every input to the 74238 Carry Look-Ahead (CLA) adder (a module of the C-7552 benchmark circuit), except the carry-in bit ($C0$), passes through a NAND and NOR gate. Every output, other than the carry-out bit, comes from a XOR gate.

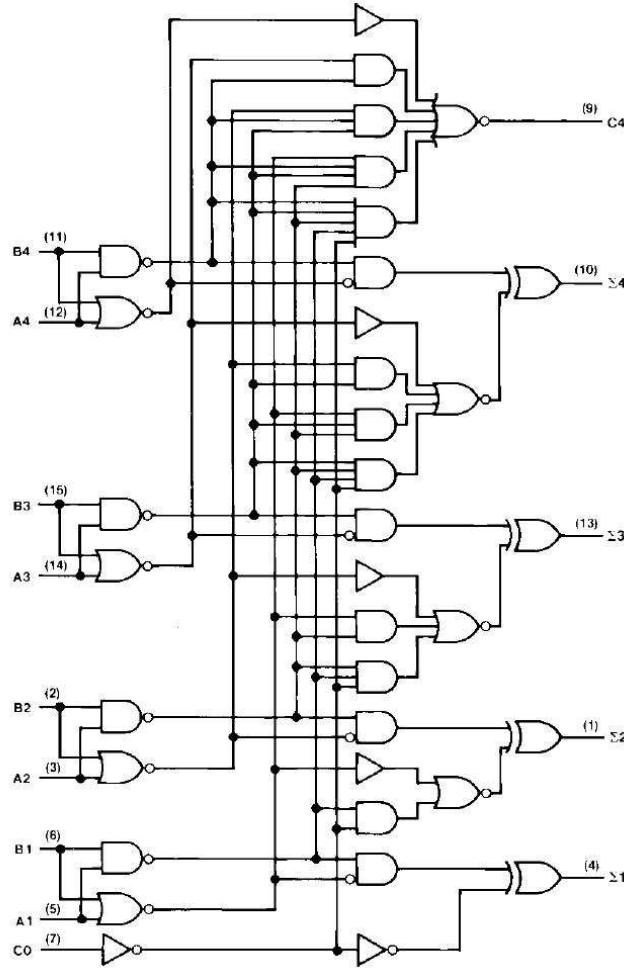


Figure 2.2: 74238 Carry Look-Ahead Adder

Identifying standard gate level designs such as these allow reverse engineers to make certain assumptions about functionality. After a more detailed inspection of the suspected adder, the reverse engineer will look for the carry look-ahead components.

Once this technique is applied to other commonly used components (i.e., comparators, multiplexers, decoders and counters), a high level model of the full design can be created that is much easier to understand. Once the functionality is identified, the circuit can be copied and used as needed. In the commercial sector, the reverse engineer can improve on the design and produce a component that may work better and cost less, since reversing often has a lower cost than the research and development of the new component. In the military sector, the reverse engineer now better understands the capability of the reversed system. This information can be used to create similar systems that may limit or negate our military advantage or decrease the time we have that advantage.

2.4 Embedded Systems

Embedded systems are commonly used today in nearly every device we use. They are found in mobile phones, credit cards, automobiles, mp3 players, computers, wrist watches, appliances, televisions, etc. The market for embedded systems has grown over the years as the demand for smaller devices with more functionality increases. This demand increases the competition to produce smaller, faster, more reliable devices. In the commercial sector, this competition creates friendly and unfriendly opportunities for reverse engineers to examine the competition's products for new and improved designs, new production techniques, or any other proprietary information which may give them an advantage. In the military sector, understanding an embedded system provides clues to weapon system functionality, cryptographic implementations and weaknesses, and ways to bypass security measures, to name but a few.

By the very fact these embedded systems are small, security problems are created. Keeping the devices small and fast, may limit the types of cryptography that can be used effectively. Many embedded devices can be easily reverse engineered if proper protection is not applied [14]. For example, the Mifare Classic Radio Frequency Identification (RFID) tokens were reverse engineered and its cipher broken.

Using knowledge of how this cipher operates, capitalizing on a random number generator implementation flaw, and using inexpensive mechanical means to expose the circuits on the chip to identify repeated gate structures the functionality of the circuit was determined. One possible defense to reverse engineering was

“Tamper-proofing can be used to protect secret keys from attackers, but provided little help against hardware reverse-engineering because the structure of the circuits will always be preserved. The implementation, however, could be obfuscated to increase the complexity of the circuit detection.”

Obfuscation will not make this approach infeasible but the degree obfuscation will increase the effort and cost to reverse engineer a circuit is unknown. The measure of circuit variant quality is determined by hiding the circuit components a reverse engineer looks for when performing this type of hardware reverse engineering.

2.5 Side channel attacks

According to Drimer [6],

“Side channel attacks rely on device-external measurable manifestations of internal processes to deduce secret data or mode of operation by exploiting the implementation rather than the algorithmic construction.”

One side channel attack method is using electromagnetic analysis (EMA). This attack relies on circuits producing magnetic fields due to the movement of electricity during internal operations of the circuit. These fields can be measured outside of the circuit using carefully tuned antennas, even without doing any physical modifications to the circuit [7]. These results show that if properly set up, the EMA attacks can be more efficient and produce better signal to noise ratios than equivalent power analysis attacks. There are two kinds of emanations, direct and unintended [1] [2]. The direct emanations are caused by current flowing through a circuit and the unintended are caused by electrical and magnetic coupling between wires and components. The main advantage of electromagnetic attacks versus power analysis attacks is electromagnetic attacks can be localized to a particular portion of a chip where the activity of interest

takes place. Also, the listening devices can be mounted on the device’s original packing [6], removing the risk of damaging the circuit in a more invasive attack.

2.6 *Invasive attacks*

Invasive attacks physically probe the device to extract information. This process removes the material that protects the metal interconnects of the circuit using chemicals and/or mechanical means, or means as complicated as using a laser cutter or focused ion beams [6]. When using this method it is necessary to have the expertise and knowledge of how the circuits are manufactured and the proper location to insert any probes. Any circuit obfuscation makes this process more difficult.

2.7 *Algorithm Overview*

A tool that aids in module or component hiding is Circuit Obfuscation via Randomization of Graphs Iteratively (CORGI) [15]. CORGI allows a designer to easily transform a circuit into a functionally equivalent, but visibly different circuit by using different selection and replacement algorithms to remove and replace logic gates in a circuit. In the case of component hiding, the original circuit C , is transformed to C' by some obfuscation function O on C , where $O(C) = C'$ and C and C' are semantically equivalent; $\forall x \in [0, 1]^n : C(x) = C'(x)$, where n is the input size of C . CORGI takes in a directed graph based representation of a given circuit and is versatile enough to perform several functions on it. The original circuit C and defines its class $C_{X-Y-S-\Omega}$, where X is the input size, Y is the output size, S is the maximum number of gates, and Ω is the basis (i.e., AND, NAND, NOR, XOR, OR, XNOR). CORGI selects a subset of the circuit’s gates, based on the defined selection strategy and makes an equivalent subset replacement based on the replacement parameter (e.g., replacing two gates with three new functionally equivalent gates). The CORGI discussion below assumes white box protection with experiments where algorithms are sequenced semantic-preserving structural transformations based on random or deterministic choices arranged in some random or deterministic manner. A circuit

obfuscator is a program that selects semantically equivalent variants of the original circuit as replacement circuits. One obfuscation measurement goal maximizes the randomness between the intermediate gates of C and its variant C' [15]. The term *gates* is used when discussing logic in combinational circuits and the term *node* is used when discussing graphs. Since the nodes in a graph represent the gate level structure of a circuit, these terms are used interchangeably.

2.7.1 Gate Selection.

2.7.1.1 Selection Strategies. There are several parameters that define how the gates of a circuit or sub-circuit are selected for replacement. These parameters may be chosen individually or chosen at random by selecting the RandomAlgorithm selection strategy. For notational purposes let Φ represent the set of all gates in the circuit, and g_x represent a gate such that $g_x \in \Phi$. The selection options defined in [15] are as follows:

- RandomSingleGate - Choose $g_1 \in \Phi$ in a random, uniform manner.
- RandomTwoGates - Choose $g_1 \in \Phi$ in a random, uniform manner. Choose $g_2 \in \Phi$ such that $g_1 \neq g_2$ and where the replacement of the sub-circuit (g_1, g_2) will not cause a cycle.
- RandomLevelTwoGates - Choose $g_1 \in \Phi$ in a random, uniform manner. Choose $g_2 \in \Phi$ such that $g_1 \neq g_2$ and where $level(g_2) = level(g_1)$ or $level(g_2) \pm 1$ and where the replacement of the sub-circuit (g_1, g_2) will not cause a cycle.
- LargestLevelTwoGates - Choose $g_1 \in \Phi$ such that $|level(g_x)| = l_{max}$ represents the maximum size of all levels within the circuit: $l_{max} = \sqcup\{|level(g_x)| \mid (g_x) \in \Phi\}$. Choose $g_2 \in \Phi$ where $g_2 \neq g_1$ and where $level(g_2) = level(g_1)$ or $level(g_2) = level(g_1) + 1$ and where the replacement of the sub-circuit (g_1, g_2) will not cause a cycle.
- OutputLevelTwoGates - Choose $g_1 \in \Phi$ where g_1 is a distinguished intermediate gate (i.e., an output of the circuit). Choose $g_2 \in \Phi$ where $g_1 \neq g_2$ and where

$level(g_2) = level(g_1) + 1$ or $level(g_2) = level(g_1)$ and where the replacement sub-circuit (g_1, g_2) will not cause a cycle.

- FixedLevelTwoGates - Choose $g_1 \in \Phi$ where, for some user-provided level criteria k , $level(g_1) = k$. Choose $g_2 \in \Phi$ where $g_2 \neq g_1$ and where $level(g_2) = level(g_1) + 1$ or $level(g_2) = level(g_1)$ and where the replacement of the sub-circuit (g_1, g_2) will not cause a cycle.
- RandomAlgorithm - Choose any selection strategy $\gamma \in S$ in a random, uniform manner. One algorithm may also weigh more heavily than others, and is done so programatically.

2.7.2 Gate Replacement. CORGI models programs as combinational Boolean circuits. Since combinational circuits do not contain feedback loops, a combinational circuit over Ω is a directed acyclic graph (DAG) having nodes that either map to a function in Ω , referred to as *gates*, having nodes in degree 0 referred to as *inputs*, and having one or more intermediate nodes as an *output*. The basis sets AND, OR, NOT, AND, NOT, OR, NOT, NAND and NOR are known to be complete, meaning any given circuit can be recreated using only the gates given the previous sets.

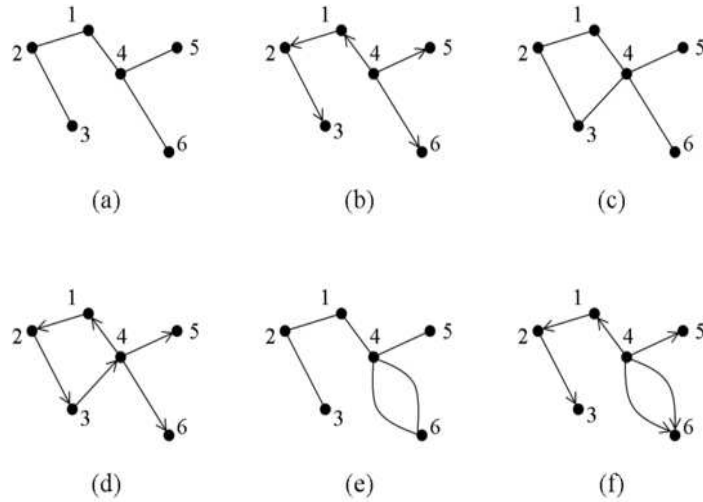


Figure 2.3: Directed and Undirected Graphs

Figure 2.3 [15] illustrates several directed and undirected graph examples. They are:

- (a) An *undirected* graph with no cycles
- (b) A *directed* graph with no cycles
- (c) A *undirected* graph with one cycle (1-2-3-4-1 and 1-4-3-2-1)
- (d) A *directed* graph with one cycle ($1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$)
- (e) An *undirected* multi-graph with one cycle (4-6)
- (f) A *directed* acyclic multi-graph with no cycles

2.7.2.1 Replacement rules. Other criteria that determine how a sub-circuit is replaced is defined by whether or not to allow the following:

- SymmetricGates - Are symmetric gates allowed? For example, are gates with inputs X_1, X_2 considered equivalent to gates with inputs X_2, X_1 ?
- AllowConstants - Are circuits granted immediate access to the constants *True* and *False*? Gates that have these properties may exist in a circuit, but it may change the properties of a set of circuits if these constants are immediately available.
- RedundantGates - Are gates allowed that are identical to other gates based on the inputs? Can there be two gates in a circuit where the truth table for each gate, based on all *circuit* inputs, is the same?
- DuplicateInputs - Are inputs to a gate allowed to originate from the same source?
- ExactCount - Does the set of replacement circuits contain all circuits with a certain size bound, or only circuits of an *exact* size? For example, if the circuit size is $S = 4$ and $\text{ExactCount} = \text{false}$, enumerating $\delta_{X-Y-S-\Omega}$ creates a family of circuits with gate sizes 1, 2, 3 and 4, where if $\text{ExactCount} = \text{true}$ the family of circuits will all have a size of 4.

- SimpleOutputs - Which gates will be defined as outputs? It may be necessary to index a circuit by an output signature and these must be restricted to specific gates. If SimpleOutputs = *true*, the outputs are then pushed to the lowest level of the graph. This option also prevents what is referred to as dangling gates; gates being considered as outputs that are never actually used.

2.7.2.2 Avoiding a cycle. As defined above, the first gate in a two gate replacement, g_1 , is chosen based on the user defined selection strategy. The second gate, g_2 , is chosen in the same manner as long as it doesn't create a cycle. An example of a possible cycle is illustrated in Figure 2.4. The two gates selected for substitution, C_{sub} , cannot be replaced without creating a cycle. The new replacement sub-circuit C_{rep} has an output that provides an input to the NAND gate C , who's output is also an input to C_{rep} . These two gates are rejected as a replacement option and another attempt is made to find a replacement that doesn't cause a cycle. Assuming C_{rep} is larger than two gates, if all possible replacements for the two gate substitution cause a cycle, the algorithm either stops executing and no replacement is made, or the algorithm replaces g_1 with a three gate equivalent. The two gate replacement case is guaranteed to work because even if no new gates are found to replace the two gates selected, the two gate selection can always be replaced with itself.

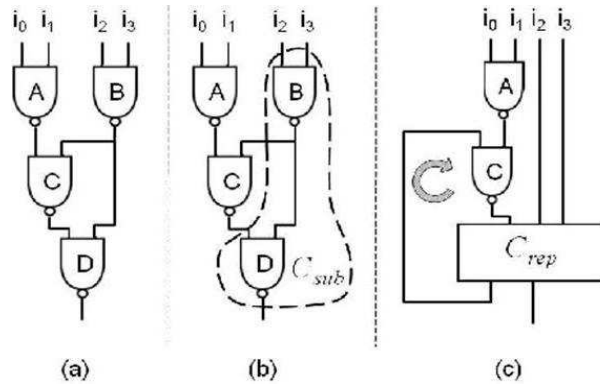


Figure 2.4: Cycle in a Replacement Circuit

Figures 2.5 and 2.6 illustrate a two gate selection with a three gate replacement. Gates 31 and 32 are selected for replacement using the RandomTwoGates algorithm.

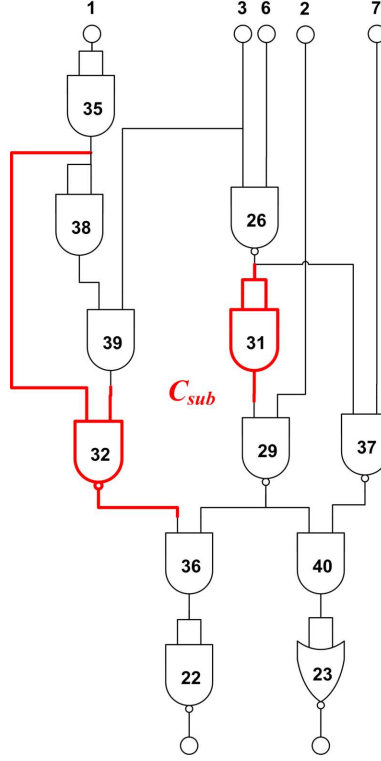


Figure 2.5: Two Gate Selection

These gates are replaced by the three gate combination formed by gates 41, 42 and 43. In Figure 2.5 there is no control flow from input 1 to the output of gate 23. After the replacement, a new control flow is introduced. This new control flow introduction is an important result useful for component hiding; it allows independent, unrelated sub-circuits to merge into one circuit.

2.8 Selection and Replacement Example

The following example illustrates the selection and replacement algorithm. Starting with the C-17 benchmark circuit in Figure 2.7, the corresponding BENCH file is created. This BENCH file is the input to the selection and replacement algorithm. In this example the RandomTwoGates algorithm is used with a two gate selection and three gate replacement strategy. Table 2.2 lists the selection and replacements performed for the first three iterations and Table 2.3 contains the original BENCH

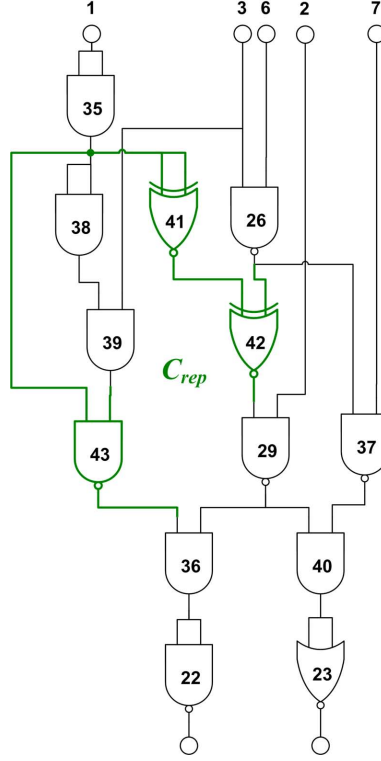


Figure 2.6: Three Gate Replacement with New Control Flow

Table 2.2: Results per Iteration for RandomTwoGate Algorithm

| Iteration | Selected Gates | Replacement Gates |
|-----------|----------------|-------------------|
| 1 | 23, 19 | 23, 24, 25 |
| 2 | 16, 10 | 26, 27, 28 |
| 3 | 27, 11 | 29, 30, 31 |

file and the BENCH files for the first three iterations. Figure 2.8 is the graph of the original BENCH file and the starting point of the algorithm. Figures 2.9-2.11 show the gate level selection and replacement for the first three iterations of the algorithm. In this example the inputs, NAND gates, and OR gates are represented by trapezoids, hexagons, and ellipses, respectively.

In the first iteration of the algorithm gates 23 and 19 are chosen for replacement. These two gates are replaced with the three gate equivalent 23, 24, and 25. Gate 24 is an identical match to gate 19. Gate 25 is a two input OR gate (buffer) and gate 23 remains the same with the output of 25 replacing the output of 19. In iteration two,

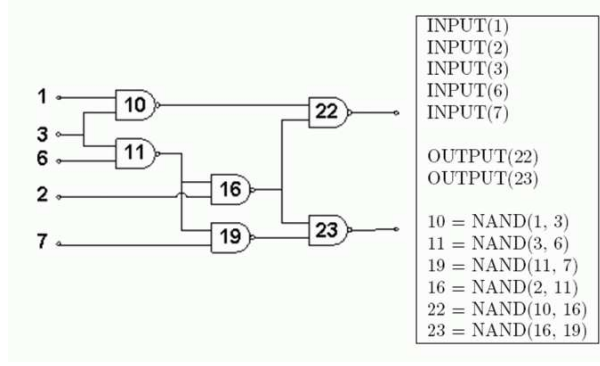


Figure 2.7: C-17 Benchmark Circuit and BENCH File

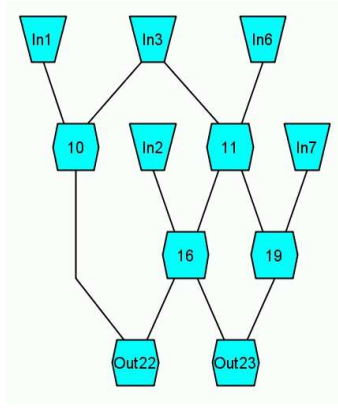
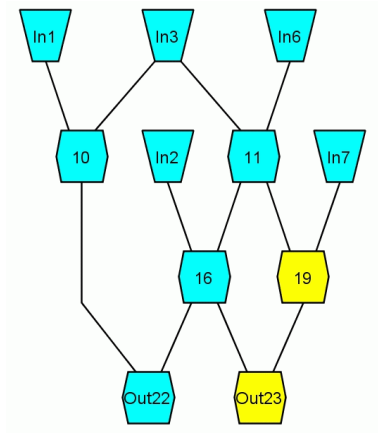


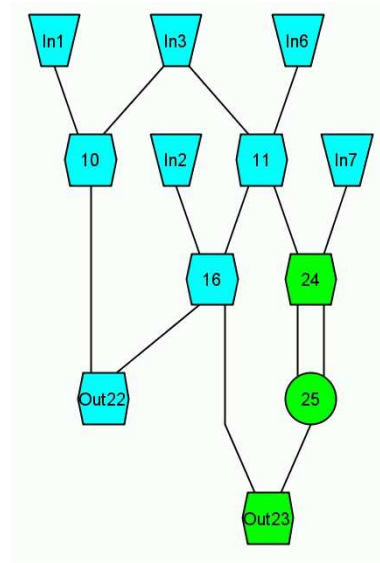
Figure 2.8: Graph Representing C-17 Benchmark Circuit

Table 2.3: Experiment BENCH Files

| Iteration 0 | Iteration 1 | Iteration 2 | Iteration 3 |
|--|---|---|--|
| INPUT(1) INPUT(2) INPUT(3) INPUT(6) INPUT(7) | INPUT(1) INPUT(2) INPUT(3) INPUT(6) INPUT(7) | INPUT(1) INPUT(2) INPUT(3) INPUT(6) INPUT(7) | INPUT(1) INPUT(2) INPUT(3) INPUT(6) INPUT(7) |
| OUTPUT(22) OUTPUT(23) | OUTPUT(22) OUTPUT(23) | OUTPUT(22) OUTPUT(23) | OUTPUT(22) OUTPUT(23) |
| 11 = NAND(3, 6) 10 = NAND(1, 3) 16 = NAND(2, 11) 19 = NAND(7, 11) 22 = NAND(10, 16) 23 = NAND(16, 19) | 11 = NAND(3, 6) 24 = NAND(7, 11) 10 = NAND(1, 3) 16 = NAND(2, 11) 25 = OR(24, 24) 22 = NAND(10, 16) 23 = NAND(16, 25) | 11 = NAND(3, 6) 24 = NAND(7, 11) 26 = OR(1, 1) 25 = OR(24, 24) 27 = NAND(2, 11) 28 = NAND(3, 26) 22 = NAND(27, 28) 23 = NAND(25, 27) | 29 = NAND(3, 6) 30 = OR(29, 29) 24 = NAND(7, 30) 26 = OR(1, 1) 25 = OR(24, 24) 28 = NAND(3, 26) 31 = NAND(2, 29) 22 = NAND(28, 31) 23 = NAND(25, 31) |

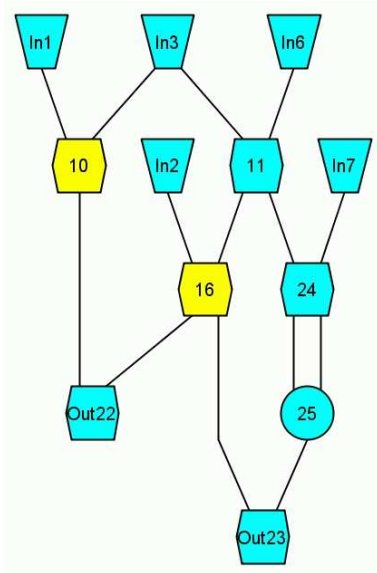


(a) Selection

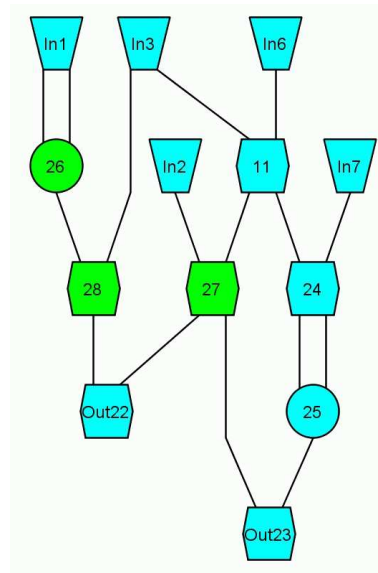


(b) Replacement

Figure 2.9: Iteration 1 Selection and Replacement
(a) and (b)

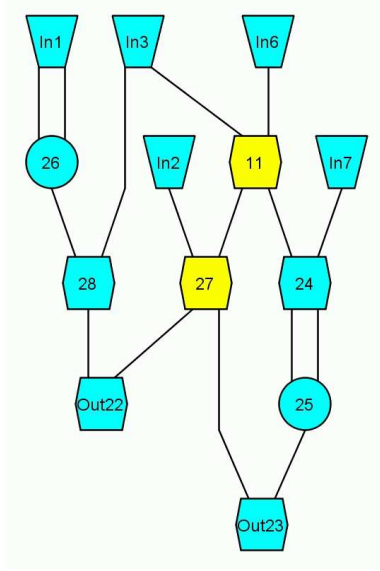


(a) Selection

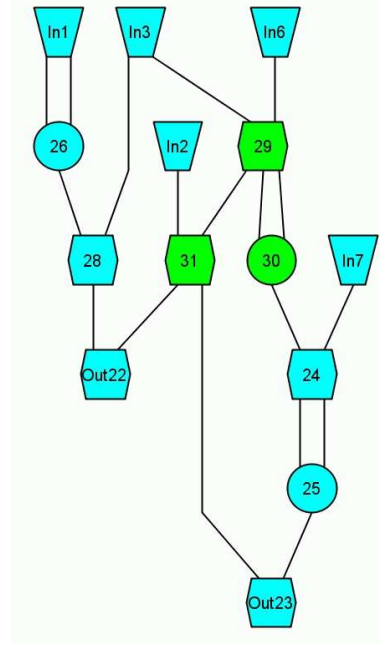


(b) Replacement

Figure 2.10: Iteration 2 Selection and Replacement
(a) and (b)



(a) Selection



(b) Replacement

Figure 2.11: Iteration 3 Selection and Replacement
(a) and (b)

gates 10 and 16 are chosen for replacement and replaced with gates 26, 27, and 28. Gates 27 and 28 provide the same functionality as 10 and 16. Gate 26 is another buffer added to the circuit. The last iteration performs the same function of introducing a new buffer to the circuit. Although this example doesn't introduce new control flow or introduce any new gates to the circuit that are not buffers, it does illustrate the basic gate selection and replacement.

2.9 Gate Level Ancestry

This section provides an introduction to gate level ancestry. An ancestor is defined by Webster [13] as a precursor of a more recent group. Thus, any new gate introduced is of direct descent from an ancestor or group of ancestors, just as a child is a direct descendant from it's parents. This is used to generate ancestry data for each gate in a circuit. Ancestry data is used to generate the colored graphs and calculate ancestral entropy, which is defined in Chapter III.

2.9.1 Ancestry File Structure. Care must be taken when creating the baseline ancestry files for experiments. The file must match the same structure as the BENCH file being used for the initial experiments. Anomalies in the colored graphs can generally be traced back to an error in the initial ancestry file. The ancestry file starts with an initial state for every identified sub-circuit or sub-component. For example, for two circuits or sub-circuits each line of the sub-component will have an assigned ancestry value of (1,0), (0,1) or (1,1). Table 2.4 provides an example of a BENCH file and the corresponding ancestry file. This file should end with ‘ancestry.txt’ for the software to perform properly. The descriptions following the # sign are not necessary but are useful in identifying the function of each gate.

Table 2.4: BENCH and Ancestry File Example

| BENCH File | Ancestry file |
|-------------------|-------------------------|
| INPUT(1) | 1(1,0,0) |
| INPUT(2) | 2(1,0,0) |
| INPUT(3) | 3(1,0,0) |
| INPUT(6) | 6(1,0,0) |
| INPUT(7) | 7(1,0,0) |
| OUTPUT(32) | 32 |
| OUTPUT(33) | 33 |
| 10 = NAND(1, 3) | 10(1,0,0) #NAND(1, 3) |
| 11 = NAND(3, 6) | 11(1,0,0) #NAND(3, 6) |
| 19 = NAND(11, 7) | 19(1,0,0) #NAND(11, 7) |
| 16 = NAND(2, 11) | 16(1,0,0) #NAND(2, 11) |
| 12 = NAND(10, 16) | 12(1,0,0) #NAND(10, 16) |
| 13 = NAND(16, 19) | 13(1,0,0) #NAND(16, 19) |
| 20 = NAND(12, 3) | 20(0,1,0) #NAND(12, 3) |
| 21 = NAND(3, 6) | 21(0,1,0) #NAND(3, 6) |
| 29 = NAND(21, 7) | 29(0,1,0) #NAND(21, 7) |
| 26 = NAND(13, 21) | 26(0,1,0) #NAND(13, 21) |
| 22 = NAND(20, 26) | 22(0,1,0) #NAND(20, 26) |
| 23 = NAND(26, 29) | 23(0,1,0) #NAND(26, 29) |
| 30 = NAND(22, 3) | 30(0,0,1) #NAND(22, 3) |
| 31 = NAND(3, 6) | 31(0,0,1) #NAND(3, 6) |
| 39 = NAND(31, 7) | 39(0,0,1) #NAND(31, 7) |
| 36 = NAND(23, 31) | 36(0,0,1) #NAND(23, 31) |
| 32 = NAND(30, 36) | 32(0,0,1) #NAND(30, 36) |
| 33 = NAND(36, 39) | 33(0,0,1) #NAND(36, 39) |
| | #end of original gates |

New nodes created by the selection and replacement algorithm are stored in a log file that contains the events of every iteration of the experiment. This log file is the input when generating ancestry files for every iteration. Each iteration the previous ancestry file is read, along with the log file, and the new ancestry values are computed and written to a new ancestry file.

2.9.2 Computing Ancestry. The ancestry value of each node is computed from the baseline ancestry file. This file is created by the user and sub-circuits are defined as required. Figure 2.12 and the ancestry file in Table 2.4 has three sub-circuits (defined as A, B, and C), each with its unique ancestry values (i.e., $A=(1,0,0)$, $B=(0,1,0)$ and $C=(0,0,1)$). These baseline ancestry values are used to generate the ancestry values of any nodes introduced into the circuit. For example, in Figure 2.13 nodes 30 and 39 from Figure 2.12 are replaced with nodes 40, 41 and 42. The ancestry value for node 41 has not changed because it has the same inputs and performs the same function as node 30 so the ancestry values are preserved.

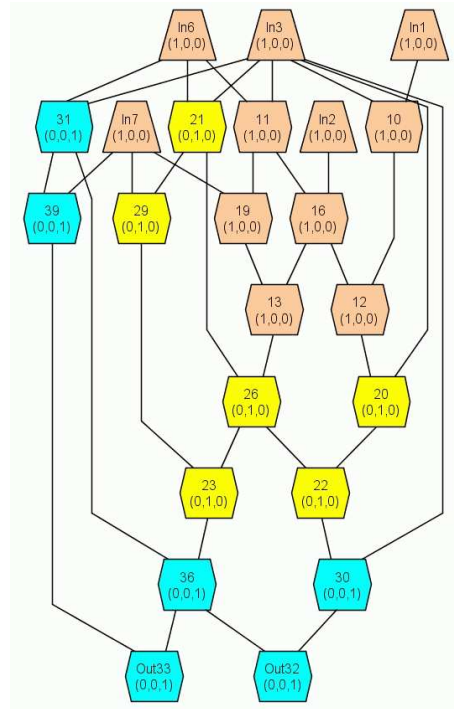


Figure 2.12: Baseline Circuit with Ancestry

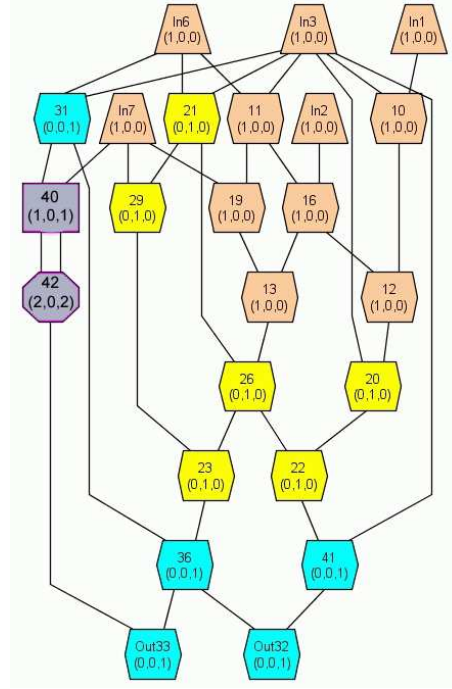


Figure 2.13: 1st Iteration

The ancestry for each gate is computed by examining the ancestry values of the parent nodes of the replaced gate. CORGI only uses two input gates, so there are only two parent nodes, X and Y , for each new node. If n is the number of sub-circuits in the baseline ancestry file, then $X = \{x_1, x_2, \dots, x_n\}$ and $Y = \{y_1, y_2, \dots, y_n\}$ are the sets of ancestry values for the parent nodes and the new ancestry set for the new node Z is $Z = \{z_1, z_2, \dots, z_n\}$ and is computed as follows:

for $i = 1$ to n

if $x_i = 0$ or $y_i = 0$

then $z_i = x_i + y_i$

if $x_i \geq y_i$

then $z_i = x_i + 1$

else $z_i = y_i + 1$

The ancestry value for node 40 is (1,0,1), meaning that there is one ancestor from circuit A, no ancestors from circuit B, and 1 ancestor from circuit C. The ancestry of node 41 is dependent on the ancestry of node 40, so the ancestry algorithm

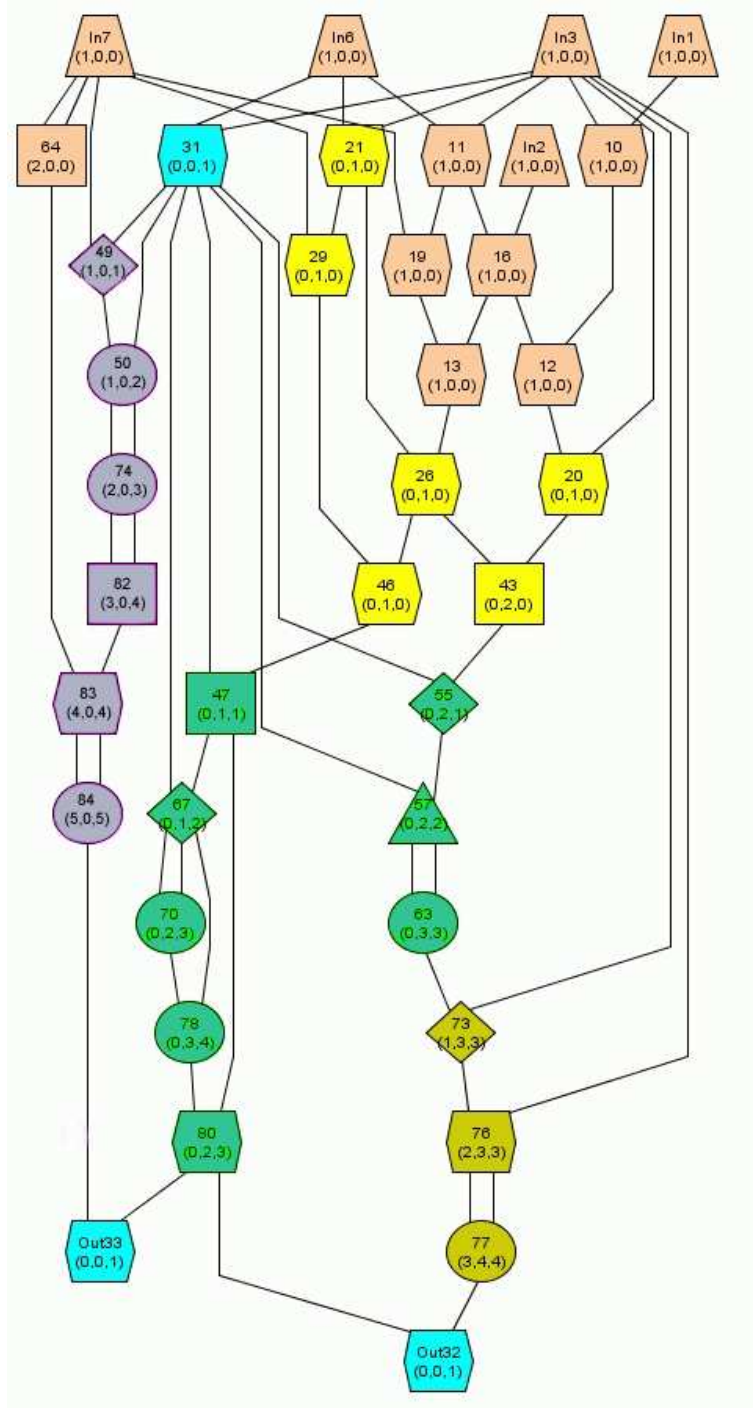


Figure 2.14: Ancestry Example Circuit, 15th Iteration

calculates the independent nodes first, then the nodes with dependencies. Node 42 has a double input from the output of node 40, so the ancestry value is (2,0,2). Figure 2.14 illustrates the introduction of gates that have ancestors from circuits A, B, and C. Nodes 73, 76, and 77 have all three ancestry values greater than zero. The parent to node 73 are nodes 63 and In3 with respective ancestry values of (0,3,3) and (1,0,0). These two values sum to (1,3,3).

2.10 Entropy

It is assumed that a circuit with nodes that have equal ancestry values from all original sub-circuits provides better obfuscation than a circuit with nodes where some sub-circuits are not represented. To evaluate this ‘goodness’, entropy measures the level of uncertainty that a specific node or gate originates from a defined component. A circuit with higher entropy is expected to be more difficult to reduce than a circuit with lower entropy. In [17], Rajgopal uses spatial entropy to measure the energy in a combinational circuit. Similarly, this research defines *ancestral entropy*, to measure the effectiveness of circuit variant producing algorithm in terms of component hiding, which is defined in Chapter III. Ancestral entropy (see Definition 5) is

$$H(P) = - \sum_{i \in N} p_i \log_2(p_i) \quad (2.1)$$

where p_i is the probability of the gate being from a defined component i . When the probability of each of the ancestry values are equal, this is the maximum entropy value, $H(P)_{max}$. The baseline ancestry value, (i.e., when only one ancestry component is greater than 0), is the minimum entropy value, $H(P)_{min}$, which equates to zero. For example, for the ancestry values (0,0,1) and (12,12,12),

$$H(P)_{min} = -1 \log_2(1) = 0$$

and

$$H(P)_{max} = -(-1/3 * \log_2(1/3) - 1/3 * \log_2(1/3) - 1/3 * \log_2(1/3)) = 1.58493$$

respectively. The $H(P)_{max}$ a tuple has equal values because that gate has an equal number of ancestors from each of the defined sub-circuits. Therefore, the gate is equally likely to have originated from any of the three original sub-circuits. This attribute provides a way to compare the transformation algorithms on like circuits. Table 2.5 contains the maximum ancestral entropy values discussed in this research.

Table 2.5: Maximum Ancestral Entropy Values

| 1 ancestor | 2 ancestors | 3 ancestors | 4 ancestors |
|------------------|------------------|----------------------|------------------|
| $H(P)_{max} = 0$ | $H(P)_{max} = 1$ | $H(P)_{max} = 1.585$ | $H(P)_{max} = 2$ |

III. Methodology

3.1 Problem Definition

When obfuscating software or hardware no effective way of measuring how well the obfuscation will perform against an attack has been developed. The level of difficulty facing the reverse engineer is a function of his experience, domain knowledge, available tools, and the software or hardware implementation.

3.1.1 Goals and Hypothesis. The goals of this research are to define component hiding, compare the circuit transformations (i.e., the output of the CORGI selection algorithms), and to develop an attribute that reflects the obfuscating properties a circuit variant provides (i.e., define an attribute for the CORGI selection algorithms to identify a circuit variant with the best component hiding properties).

We hypothesize the iterative selection and replacement algorithms provide measurable component hiding properties, resulting in the circuit transformations not easily undone by a circuit reducer (or a reverse engineer). A polymorphic circuit engine based on iterative selection and replacement (CORGI) alters circuits to hide circuit or sub-circuit components. Component hiding, defined later, adds difficulty to the reverse engineering effort by preventing a reverse engineer from grouping gates into a higher level functional model, thus reducing the amount of black box analysis. CORGI provides a graph of each iteration when creating a circuit variant. This graph shows circuit changes, but doesn't provide sufficient visual information on how the changes are occurring. Creating an ancestry value for each gate in the circuit measures how much each circuit changes during each iteration and it provides a history for each node, as well as a colored graph as a visual representation. Using CORGI to select and replace gates in a circuit and comparing this circuit variant to the original provides information about the quality of a circuit variant based on the different selection algorithms used. Entropy measures the uncertainty of whether a gate belongs to an original component.

3.1.2 Approach. Baseline circuits and configurations are identified (i.e., series, parallel, shared inputs, etc.), as well as the parameters CORGI uses to create the circuit variant. Experiments on series and parallel circuits using different CORGI selection algorithms identifies the contribution of each algorithm to the creation of a circuit variant. Ancestry information and changes to each baseline circuit are recorded at every iteration providing a history of gate level transformations.

3.2 *System boundaries*

The System Under Test (SUT) is the Circuit Transformation System (CORGI) shown in Figure 3.1. The specific components under test are the following selection and replacement algorithms:

1. RandomTwoGates selection and replacement algorithm
2. RandomLevelTwoGates selection and replacement algorithm
3. LargestLevelTwoGates selection and replacement algorithm
4. OutputLevelTwoGates selection and replacement algorithm
5. FixedLevelTwoGates selection and replacement algorithm
6. RandomAlgorithm selection and replacement algorithm

These algorithms are applied to each of the series and parallel circuits described in this section. A comparison of the resulting circuit variant determines the obfuscation effectiveness of the selection algorithm for that circuit.

3.3 *System Services*

The service provided by the system is the selection and replacement of logic gates based on the system parameters. Changes to the circuit are recorded in a graph file, a bench file, a log file, and an ancestry file. The first possible system output are the files representing a functionally equivalent circuit variant. A second possible

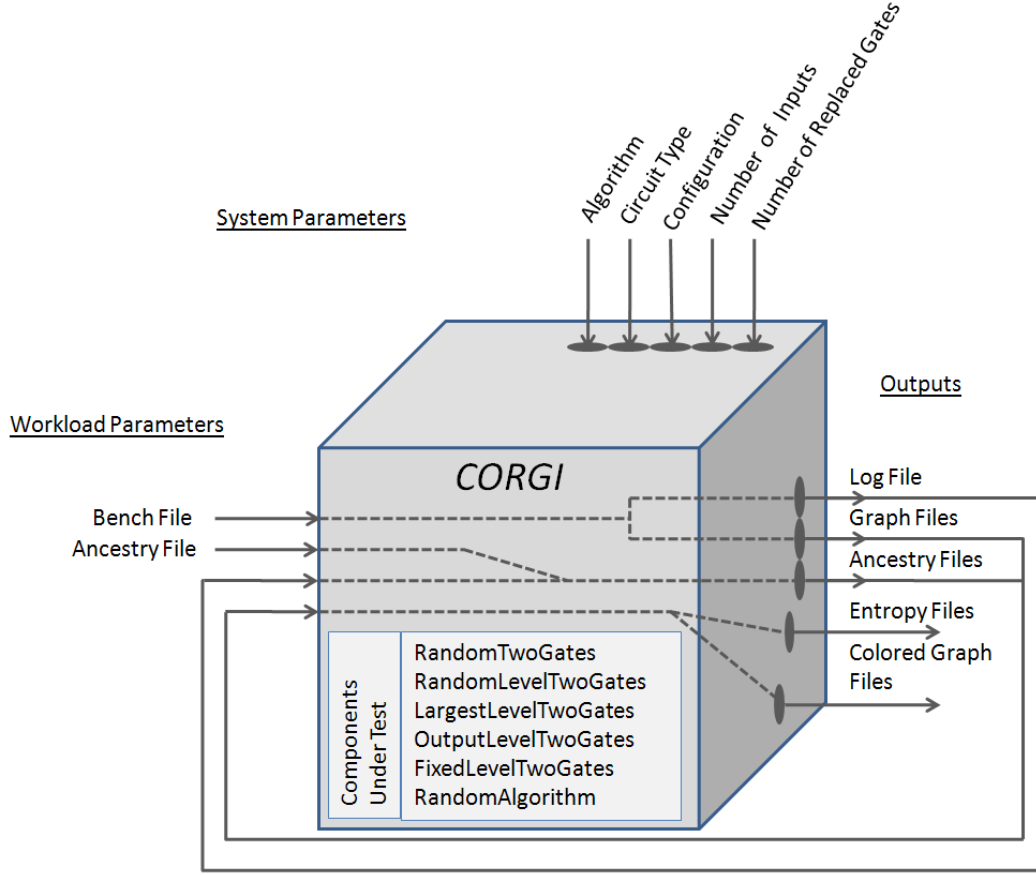


Figure 3.1: Circuit Transformation System (CORGI)

output are files representing a circuit variant that is not functionally equivalent. The last possibility is that nothing occurs, signifying a failure of the system.

3.4 Workload

The workload of this system is three tiered. The first workload is a circuit bench file that specifies the circuit or sub-circuits and their configuration. This may be a stand alone circuit or sub-circuits connected in series or parallel with independent or shared inputs. CORGI accepts bench files as input for the gate selection algorithms and provides a bench file and a graph file as an output for each iteration, along with a log file that stores the details of each iteration. This log file, along with a user provided ancestry file, is the second workload and generates ancestry information for

each iteration of the selection and replacement algorithm. The third workload is the previously generated graph and ancestry files for each iteration. These files are used to produce the final colored graph that visually displays the ancestry of each gate in the circuit and file that contains the ancestral entropy calculation for each node, the average ancestral entropy at each level and at the output of the circuit. Figure 3.1 illustrates how the workload and output are related.

3.5 System Parameters

The system parameters for this research are as follows:

1. Selection and Replacement Algorithm

- RandomTwoGates
- RandomLevelTwoGates
- LargestLevelTwoGates
- OutputLevelTwoGates
- FixedLevelTwoGates
- RandomAlgorithm

2. Circuit Type

- C-17 Benchmark Circuit

3. Circuit Configurations

- Series Configuration
- Parallel Configurations

4. Number of gates replaced

- Replace Two Gates With 3 gates
- Replace Two Gates With 4 gates

The C-17 benchmark circuit, shown in Figure 2.7 is used because it is a small circuit composed of NAND gates and is part of the ISCAS '85 suite of benchmark circuits. This circuit is representative of the many small sub-circuits found in larger circuits, such as adders, multiplexors, and decoders. This circuit is examined in four different series and parallel configurations. Some of the configurations use independent inputs to each circuit, while others share some, or all, of the inputs. Figures 3.2 and 3.3 illustrate the four C-17 circuit configurations.

The six different selection and replacement algorithms are tested to determine if any one algorithm outperforms another, in terms of obfuscation properties. Multiple circuit types are chosen to see if different algorithms perform better on series or parallel circuits with different configurations. Using the replacement parameters of three and four gates test whether one type of replacement is better than another. Smart selection, in the case of the RandomTwoGates replacement algorithm, means at least one original node is used in every selection and replacement iteration until all original nodes have been replaced at least once prior to any two new nodes being selected for replacement. Allowing duplicate inputs and redundant gates provides easier gate replacement and improves the chances independent circuits will have gates selected for replacement, allowing earlier merging of circuits.

3.6 Performance Metrics

Three different performance metrics are used to measure obfuscation. The first is the number of circuits or sub-circuits removed from the original circuit based on the calculated ancestry information. The colored graph shows the gate ancestry and new colors represent gates that have an ancestor from different circuits (defined in the original ancestry file). The more original sub-circuits, or components of a circuit removed and new circuits introduced, the better the obfuscation. The second measurement is the percentage of original gates remaining in a circuit. A lower percentage of original gates remaining in the final circuit variant correlates with a better level of obfuscation. The final measurement calculates the ancestral entropy of every node,

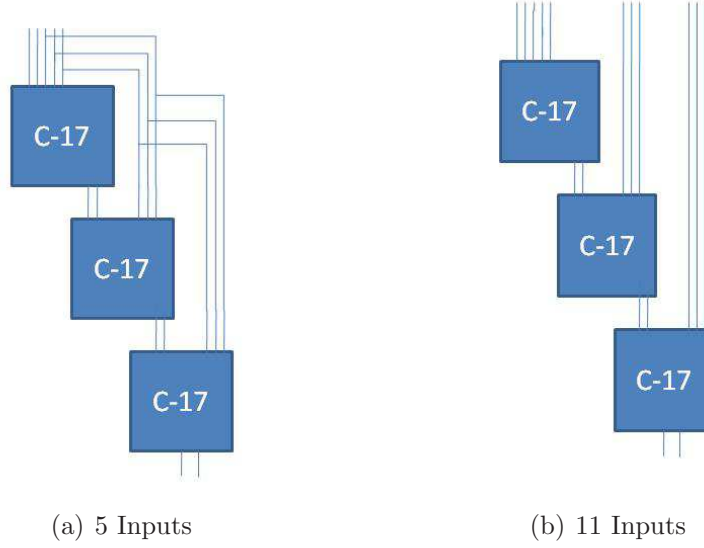


Figure 3.2: C17 Series Circuits (a) and (b)

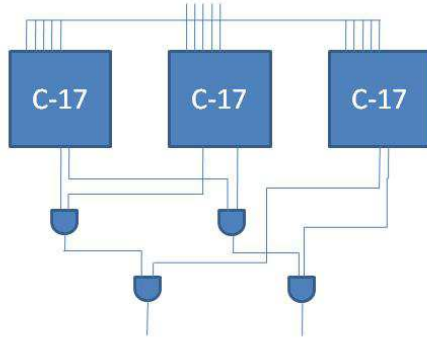
each level, and outputs of the circuit. Measuring the uncertainty of the ancestry provides an attribute for comparing the effectiveness of obfuscation each of the circuit variants provide.

3.7 Factors

Table 3.1 contains the factors and levels for this research effort. The six selection and replacement algorithms are the experiment factors. Each algorithm uses five different circuits and performs a two gate selection with a three and four gate replacement. This results in 60 experiments, which are performed three times each.

3.8 Evaluation Technique

CORGI is used to simulate a random walk from the original circuit to the final circuit variant. CORGI uses SmartSelectionAlgorithm.java program for each iteration of the experiment. This code allows the user to specify the type of selection and replacement algorithm, the number of iterations to perform, the number of experiments to repeat, whether to allow duplicate input gates and redundant gates, the number



(a) Shared Inputs



(b) Independent Inputs

Figure 3.3: C-17 Parallel Circuits (a) and (b)

Table 3.1: Factors and Levels

| Factor | Level | | | |
|----------------------|--------------|---------------|-------------|----------------|
| Algorithm | Circuit Type | Configuration | Inputs | Gates Replaced |
| RandomTwoGates | C-17 | Series | 5 | 3 |
| RandomLevelTwoGates | | | | 4 |
| OutputLevelTwoGates | | | 5 'Split' | 3 |
| FixedLevelTwoGates | | | | 4 |
| LargestLevelTwoGates | | | 11 | 3 |
| RandomAlgorithm | | | | 4 |
| | | Parallel | Shared | 3 |
| | | | | 4 |
| | | | Independent | 3 |
| | | | | 4 |

of gates to select, the number of replacements and many other parameters. A log file and graph files are produced that are used later. The flow chart in Figure 3.4 shows which files are needed to create subsequent files. The selection and replacement algorithm uses the baseline BENCH file to generate a BENCH file and graphml file for every iteration of the experiment. A log file that records the operations performs during every iteration. The baseline ancestry file is created from the baseline BENCH file and, along with the log file, creates the ancestry files for every iteration. To color the graphs the graphml files and the ancestry files generate new graphml files that

provide color, shape, number, and ancestry information for each node in the circuit at every iteration. Entropy is computed using the BENCH files and the ancestry files. Ancestral entropy is calculated for each node, at the output of the circuit, and at every level of the circuit.

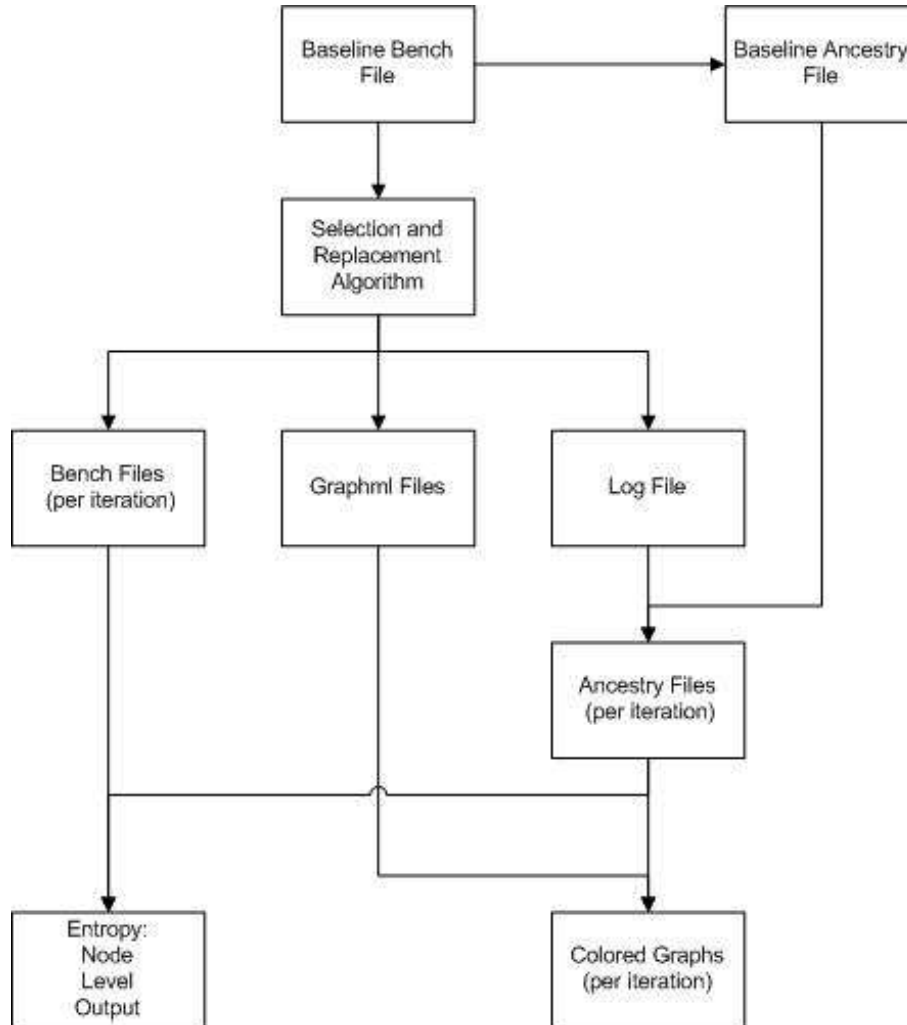


Figure 3.4: File Creation Flow Chart

The ColorGraph.java program uses the ancestry files, along with the original graph files, to create a colored graph for every iteration that visually represents how gates are related to one another as well as a file that identifies when new sub-circuits are introduced or removed, based on their ancestry. These graphs identify any sub-circuits removed, any trends for new gates introduced and the percentage of gates no

longer associated with a single sub-circuit. The GetEntropy.java program also uses the ancestry files to compute the ancestral entropy of each gate in the circuit, the average ancestral entropy at each circuit level and at the outputs.

3.9 Experimental Design

The replacement rules discussed in Section 2.7.2.1 for three gate replacement experiments are set to *false* except DuplicateInputs and RedundantGates. This setting allows for the introduction of new circuit control flow when executing a three gate replacement. However, many new gates introduced act as buffers (e.g., two input *OR* and *AND* gates). The introduction of the new control flow will allow many of these buffers to be replaced in future circuit variants. These two settings are set to *false* in the four gate replacement experiments because new control flows are still introduced.

Initial experiments indicate there is no benefit to performing a two gate selection with a two gate replacement on independent parallel circuits or series circuits as there is no increase in entropy. When duplicate inputs and redundant gates were disallowed independent circuits did not merge. Every algorithm, except RandomTwoGates, did not change the structure of the circuits or introduce any new gates. Figure 3.5 contains the results for a two gate replacement strategy on a parallel circuit for all of the selection and replacement algorithms except RandomTwoGates. The RandomTwoGates experiments did introduce new matching gates. But, since no change resulted to the physical layout of the circuits it provided no component hiding characteristics. Figure 3.6 displays the same results for a series circuit. Although these gates were replaced several times over in both figures, the circuit structure remains identical.

3.10 Adding Color and Shape to each Gate

To visualize the circuit transformations, the graphs generated by the chosen algorithm incorporate colors for circuit or sub-circuit identification and shapes for gate operation. These colors and shapes can be easily changed in the source code.

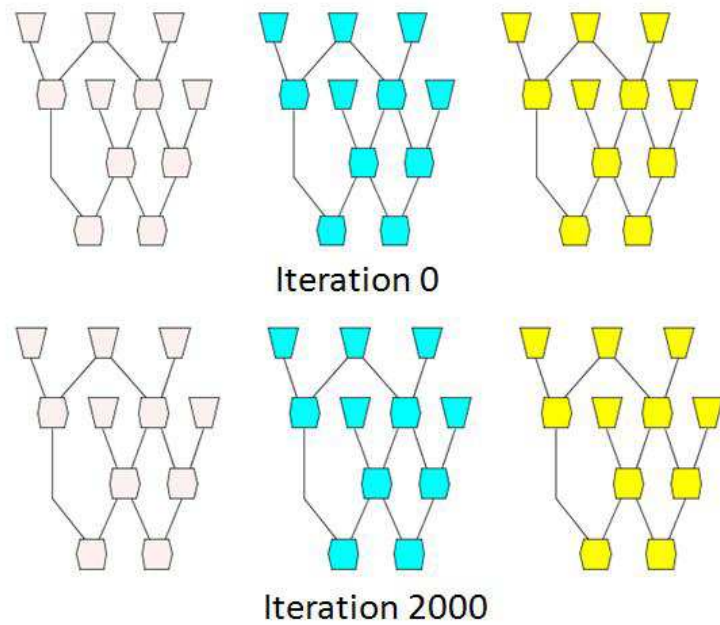


Figure 3.5: C-17 Parallel Circuit Maintaining Structure

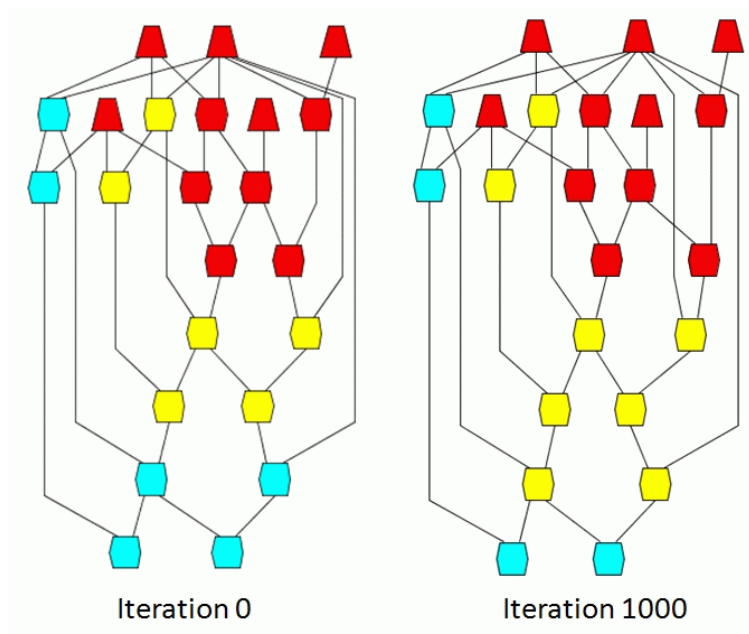


Figure 3.6: C-17 Series Circuit Maintaining Structure

Table 3.2: Gate Operation and Matching Shape

| Gate Operation | Node Shape |
|----------------|------------|
| AND | Rectangle |
| NAND | Hexagon |
| OR | Ellipse |
| NOR | Octagon |
| XOR | Triangle |
| XNOR | Diamond |

Table 3.3: Coloring Schemes

| Ancestry 1 | Ancestry 2 | Sum |
|--------------|------------|-----|
| First Level | | |
| 001 | 010 | 011 |
| 001 | 100 | 101 |
| 010 | 100 | 110 |
| Second Level | | |
| 001 | 011 | 011 |
| 001 | 101 | 101 |
| 001 | 110 | 111 |
| 010 | 011 | 011 |
| 010 | 101 | 111 |
| 010 | 111 | 111 |
| 100 | 011 | 111 |
| 100 | 101 | 101 |
| 100 | 111 | 111 |
| Third Level | | |
| 011 | 101 | 111 |
| 011 | 110 | 111 |
| 110 | 101 | 111 |

and (1,1,0). With all three combinations of addition the possible results are (0,1,1), (1,1,1), and (1,1,0). The uniqueness of each gate is not properly represented by a color if the goal is to identify every level of ancestry. When (0,1,0) and (1,1,0) are summed, the result for determining color is (1,1,0), which is the assigned color value for an original sub-circuit. Choosing the initial values of (0,0,1), (0,1,0), and (1,0,0) will provide correct coloring for the first level, but fails for much of the second level. Table 3.3 and Figure 3.8 illustrate examples of this. The first level is the possible combinations of addition for the three original ancestry values. The second level is

the possible combinations of one of the original values summed with the results from the first level. The third level shows the addition combinations of the results from the first level. By the third level all of the resulting sums are the same. Color values containing all ones are the nodes with ancestors from every pre-defined component.

Different schemes may allow more children nodes to be colored differently than both parent nodes before running out of options. With a three sub-circuit ancestry there can be only 7 colors in the graph (i.e., binary 000 to 111). It is not recommended to use (0,0,0) since this does nothing under addition and will not result in a color change for newly introduced nodes. It is possible to use a four or five digit ancestry scheme while only having 2 or 3 identified sub-circuits. This provides up to 32 colors to identify sub-circuits. The software currently allows the use of 15 colors, but can be easily modified. This provides more options but the same problems in Table 3.3 apply.

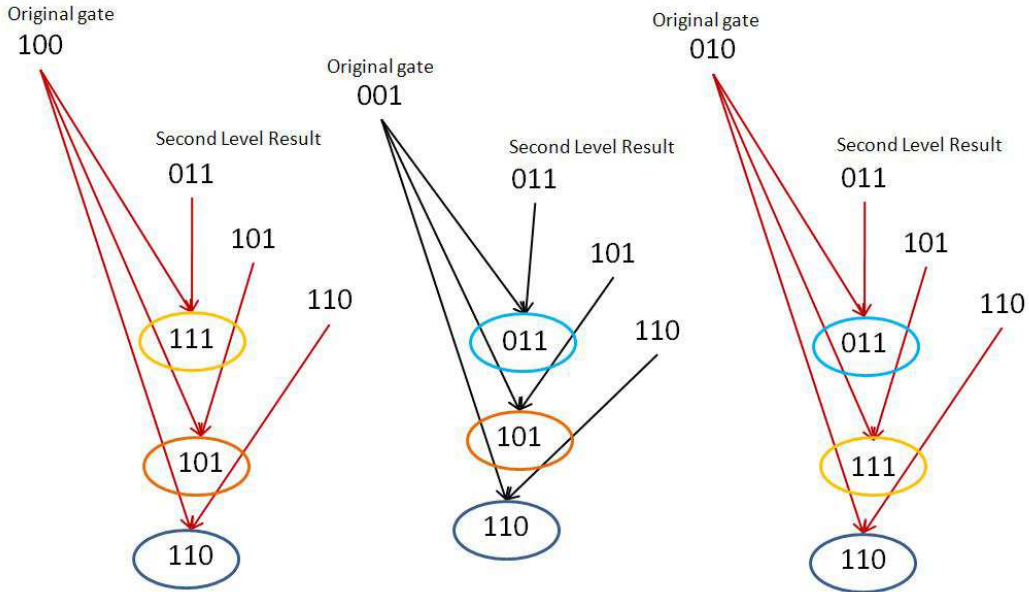


Figure 3.8: Matching Results for 3 Levels

3.11 Component Hiding

To properly describe component hiding the following definitions are provided:

Definition 1. *Circuit*: A directed, acyclic graph $D(V, E)$ of inputs, logic gates and outputs where 1) V is the set of nodes of the circuit where each node is either an input with an assigned value or a gate with a corresponding Boolean function $f : \{0, 1\}^x \{0, 1\} \rightarrow \{0, 1\}$, 2) E is the set of wires that connect nodes, 3) the gate set G of a circuit is the set of all gate nodes within V , 4) inputs are nodes in the graph with no fan-in and 5) outputs are distinguished nodes.

Definition 2. *Component*: Given a gate set G and an input set I of a circuit P and an integer $k > 1$, where k is the number of components, a set C of components $\{c_1, \dots, c_k\}$ partitions G and I into k disjoint sets of inputs and/or gates.

Definition 3. *Component Hiding*: Given 1) an original circuit P with a gate set G and input set I and 2) a set C which divides G and I into k disjoint components $\{c_1, c_2, \dots, c_k\}$, any functionally equivalent variant P' of P with gate set G' accomplishes component hiding if and only if, for all gates $g \in G'$, maximal uncertainty exists regarding the relationship of any gate g to any original component $\{c_1, c_2, \dots, c_k\}$.

Definition 4. *Ancestry*: Given m number of gates in a circuit, k number of defined components in a circuit, and a tuple $T = \{t_1, t_2, \dots, t_{k-1}, t_k\}$ assigned to each gate g_m in a circuit, where t describes the composition of g_m in terms of the n^{th} component.

Definition 5. *Ancestral Entropy*: Let $P = (p_1, \dots, p_n)$ be a probability distribution on the set of $N = (1, \dots, n)$ components in a circuit. The entropy of P is the function $H(P) = - \sum_{i \in N} p_i \log_2(p_i)$, where N is the number of components in a circuit and p_i is the probability of a gate being from a defined component i . Maximum ancestral entropy, $H(P)_{\max}$, is achieved when $p_i = p_{i+1} = \dots = p_{n-1} = p_n$. $H(P)_{\min}$ is achieved when $p_i = 1$.

There are four cases to consider when discussing component hiding. They are listed below using Figure 3.9 as an example.

- Case 1. Input from the circuit boundary and output to a component: Component A receives the inputs from the circuit boundary and its output is connected to the internal component B .
- Case 2. Input from a component and output to a component: Internal component B receives the inputs from component A and its output is connected to component C .
- Case 3. Input from a component and output to the circuit boundary: Component C receives its inputs from internal component B and provides output to the circuit boundary.
- Case 4. Input from the circuit boundary and output to the circuit boundary: Component D receives the inputs from the circuit boundary and provides output to the circuit boundary.

To measure the level of component hiding, GetAncestry.java uses the log file and an ancestry file that identifies sub-circuits or sub-components of the initial circuit to generate ancestry files for every iteration in the experiment. This ancestry information is used to measure the uncertainty a node in the circuit belongs to any previously identified component.

3.12 Methodology Summary

The main goals of this research are to define component hiding, compare the CORGI selection and replacement algorithms, identify obfuscating properties, and define an attribute that reflects these properties. A baseline circuit is used (i.e., C-17 sub-circuits in series and parallel configurations) as input to the six selection algorithms and an entropy-based calculation is used to define an attribute that reflects the component hiding properties of each algorithm.

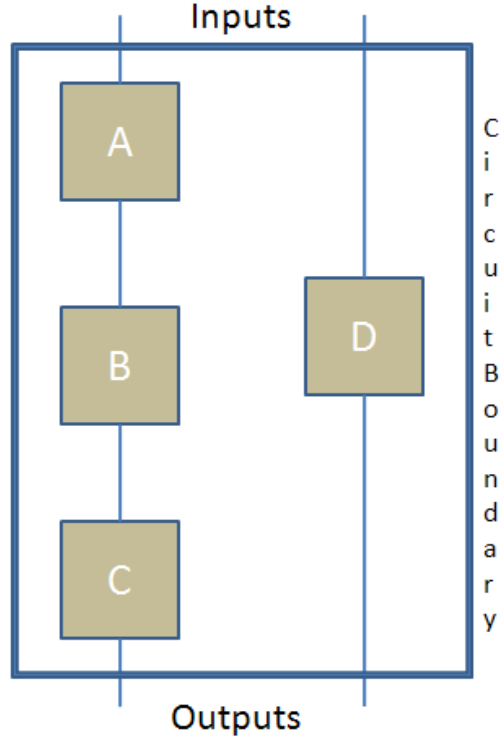


Figure 3.9: Component Hiding Cases

The factors studied are six CORGI selection and replacement algorithms; RandomTwoGates, RandomLevelTwoGates, RandomAlgorithm, LargestLevelTwoGates, FixedLevelTwoGates, and OutputLevelTwoGates. The input to the algorithms are series and parallel circuits in BENCH format and the output is ancestry files, BENCH files, entropy files, and colored graph files. These resulting files are evaluated and compared to determine which algorithm creates circuit variants with better component hiding properties for a given workload.

IV. Results

Component hiding is an important operation that can complicate reverse engineering efforts. If sub-circuits are hidden from a reverse engineer he cannot group logic gates into a higher level model, forcing him to do a black box analysis of the circuit. The following experiments show how the different CORGI selection and replacement algorithms perform. Ancestral node entropy, circuit level entropy and output level entropy values are compared to determine which algorithms provide better component hiding properties using different types of circuits. This chapter is divided into the following sections: Section 4.1 discusses three gate replacement series circuits, Section 4.2 discusses three gate replacement parallel circuits, Section 4.3 describes the results of reducing three and four gate replacement circuits, Section 4.4 and 4.5 discuss the results of the four gate replacement series and parallel circuits respectively and Section 4.6 discusses validation.

4.1 *Three Gate Replacement C-17 Series Circuits*

Two series circuit configurations are used to test each of the six algorithms. The results are compared and an algorithm is chosen that creates the best circuit variants with obfuscating properties. The operation of each algorithm is explained in Section 2.7.1.1, but is summarized here as well. The two high level series designs used in this research are illustrated again in Figure 4.1.

The five input series circuit shown in Figure 4.1(a) are a series of three C-17 circuits with shared inputs. This circuit is used in two different experiments for each algorithm. CORGI selection algorithms often allow a sub-circuit(s) to be completely replaced by the new gates introduced to the circuit. The removal of a circuit affects the entropy calculation, and is explained below. This first C-17 sub-circuit in each case has five inputs and two outputs. Two outputs and three of the initial inputs serve as inputs to the second sub-circuit. The same holds true for the final C-17 sub-circuit. The only difference in the two experiments is how the ancestry file is created. Shown in Table 4.1, Ancestry File A connects all of the inputs to the first C-17 sub-circuit.

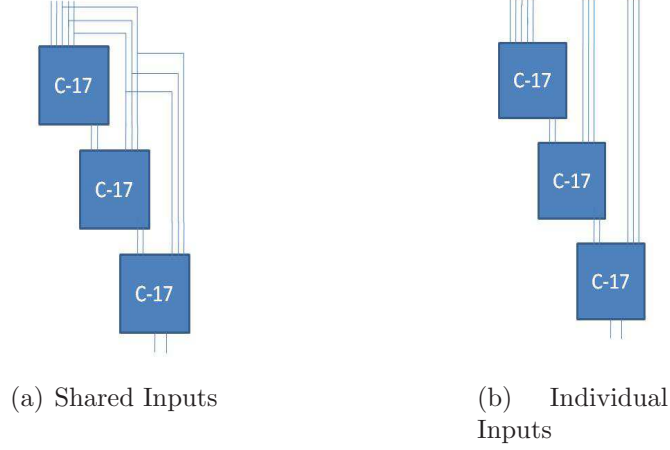


Figure 4.1: High Level C-17 Series Circuits
(a) and (b)

The Ancestry File B connects two inputs to the first C-17 sub-circuit, two inputs to the second C-17 sub-circuit, and one input to the last C-17 sub-circuit. The only change required in the ancestry file is the assignment of ancestry to the input values. This circuit is referred to as the five ‘split’ input circuit. These input configurations are illustrated in Figure 4.2.

The experiments in Sections 4.1.1-4.1.6 select two gates for replacement and performs a three gate equivalent replacement on the series circuits. Sections 4.2.1-4.2.6 contains the parallel circuit experiments. The results of these experiments are summarized in Section 4.1.7 and 4.2.7, respectively.

4.1.1 *FixedLevelTwoGates Experiment.*

4.1.1.1 Five Inputs Circuit Experiment. The fixed level algorithm selects the first gate for replacement from a pre-defined level k . The second gate is either chosen from level k or level $k + 1$. In this experiment, k is set to level one resulting in all replacements being made at level one or level two. Level zero is the output level. These replacements cause gates to be added to the k^{th} level of the graph [15]. As illustrated in Figure A.1, the graph becomes tall and narrow, while preserving the upper and lower portions of the graph (i.e., the levels above

Table 4.1: C-17 Series Experiments Ancestry Files

| Ancestry File A | Ancestry File B |
|-------------------------|-------------------------|
| #INPUTS | #INPUTS |
| 1(1,0,0) | 1(1,0,0) |
| 2(1,0,0) | 2(1,0,0) |
| 3(1,0,0) | 3(0,1,0) |
| 6(1,0,0) | 6(0,1,0) |
| 7(1,0,0) | 7(0,0,1) |
| #OUTPUTS | #OUTPUTS |
| 32 | 32 |
| 33 | 33 |
| 10(1,0,0) #NAND(1, 3) | 10(1,0,0) #NAND(1, 3) |
| 11(1,0,0) #NAND(3, 6) | 11(1,0,0) #NAND(3, 6) |
| 19(1,0,0) #NAND(11, 7) | 19(1,0,0) #NAND(11, 7) |
| 16(1,0,0) #NAND(2, 11) | 16(1,0,0) #NAND(2, 11) |
| 12(1,0,0) #NAND(10, 16) | 12(1,0,0) #NAND(10, 16) |
| 13(1,0,0) #NAND(16, 19) | 13(1,0,0) #NAND(16, 19) |
| 20(0,1,0) #NAND(12, 3) | 20(0,1,0) #NAND(12, 3) |
| 21(0,1,0) #NAND(3, 6) | 21(0,1,0) #NAND(3, 6) |
| 29(0,1,0) #NAND(21, 7) | 29(0,1,0) #NAND(21, 7) |
| 26(0,1,0) #NAND(13, 21) | 26(0,1,0) #NAND(13, 21) |
| 22(0,1,0) #NAND(20, 26) | 22(0,1,0) #NAND(20, 26) |
| 23(0,1,0) #NAND(26, 29) | 23(0,1,0) #NAND(26, 29) |
| 30(0,0,1) #NAND(22, 3) | 30(0,0,1) #NAND(22, 3) |
| 31(0,0,1) #NAND(3, 6) | 31(0,0,1) #NAND(3, 6) |
| 39(0,0,1) #NAND(31, 7) | 39(0,0,1) #NAND(31, 7) |
| 36(0,0,1) #NAND(23, 31) | 36(0,0,1) #NAND(23, 31) |
| 32(0,0,1) #NAND(30, 36) | 32(0,0,1) #NAND(30, 36) |
| 33(0,0,1) #NAND(36, 39) | 33(0,0,1) #NAND(36, 39) |
| #end of original gates | #end of original gates |

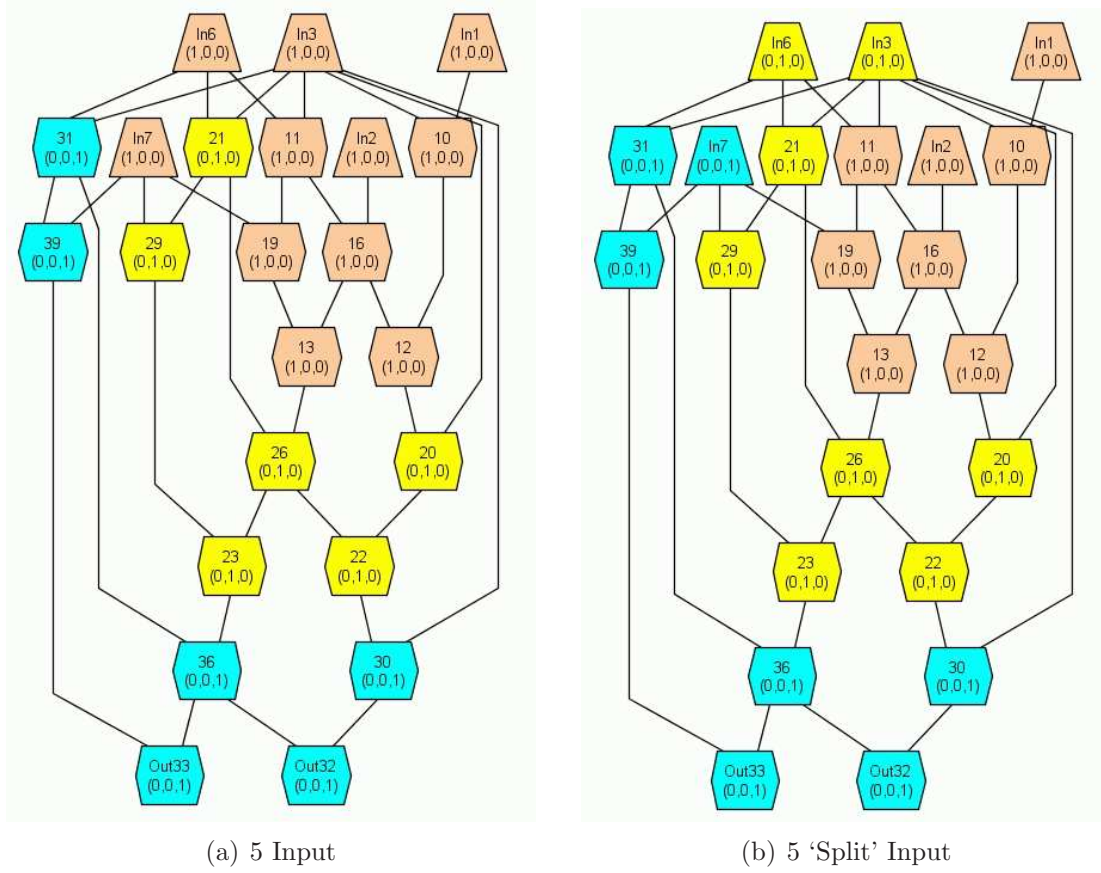


Figure 4.2: Fixed Level C-17, 5 Input Circuits
(a) and (b)

and below the fixed level and one level higher). Notice the circuit outputs and the upper portion of the graphs do not change. If the fixed level is selected to be level zero, this algorithm performs the same function as the output level algorithm. For all fixed level experiments, the original gates are not replaced when creating the circuit variants. This can be seen in Figure A.1 as early as iteration 50. The upper portion of the graph maintains its structure, preserving two of the three original circuits. Because the fixed level is set to one for this experiment, outputs are *never* selected for replacement, guaranteeing they remain part of the original circuit.

Illustrated in Figure A.2(a), the average circuit level ancestral entropy reaches $H(P)_{max}$ in levels 1 to 309. The entropy values above level 309 represent most of the original circuit and therefore, have lower entropy values. Because the gates to be

replaced are selected in levels one and two, these levels of the graph quickly reach, or nearly reach $H(P)_{max}$ causing most of the nodes in the 1000th iteration to have high entropy values. The average output level ancestral entropy in Figure A.2(b) maintains a zero value because the outputs are never replaced. Of the 1023 nodes 973 have ancestors from all three sub-circuits, 966 of these have an ancestral entropy value of 1.569 or higher which is 99% of $H(P)_{max}$. However, much of the original circuit has not been replaced at all. The original circuit has 18 nodes (all with $H(P)_{min}$ value) while the 1000th iteration has 15 nodes that have a $H(P)_{min}$ value.

4.1.1.2 Five ‘Split’ Inputs Circuit Experiment. This circuit configuration is identical to the five input circuit, except the ancestry file connects the inputs to each of the sub-circuits. The graphs in Figure A.3 are structurally the same as the five input graphs. However, the extrusion of gates has a $H(P)_{max} = 1$, not 1.585 because of the way the ancestry file connects the inputs to each sub-circuit. The upper rightmost sub-circuit remains unchanged resulting in the new nodes added to the graph only having two non-zero ancestry values. In other words, each new node only has ancestors from two of the three original circuits.

Examining Figure A.4, the average circuit level ancestral entropy is .997 or greater from level one to level 324. This highlights the extrusion of gates and produces gates with ancestors from two of the original three sub-circuits. The average output ancestral entropy is still zero because the output nodes are still part of an original sub-circuit. Of the nodes in the circuit, 97.36% have ancestors from two of the three subcircuits, while no nodes have ancestors from all three sub-circuits. The remaining nodes are direct descendents of an original sub-circuit, while 14 of the original 18 gates remain in the circuit.

4.1.1.3 Eleven Inputs Experiment. In this circuit configuration the sub-circuit share no inputs but to keep them in series the output of the first circuit feeds the inputs for the second, and the output of the second circuit feeds the input of the third circuit. This configuration is shown in Figure A.5(a) clearly isolates each

of the sub-circuits. The average circuit level entropy has a maximum value equal to one because the the uppermost left circuit doesn't become part of any circuit variant, resulting in all new gates having a maximum of two non-zero ancestry values. Again, the output nodes are never modified so the average output ancestral entropy is equal to zero. Also, 96% of the nodes in the circuit have ancestors from two of the original sub-circuits, while no nodes have ancestors from all three sub-circuits. At the 1000th iteration, 15 of the 18 gates still remain in the circuit. These results are similar to the five 'split' input experiment as Figure A.6 displays.

4.1.2 OutputLevelTwoGates. The output level algorithm selects the first gate for replacement from the output level (i.e., level zero) and the second from either level zero or level one and makes an equivalent three gate replacement. This algorithm performs very similar to the fixed level algorithm when the fixed level equals one, except in this case the output gates are replaced.

4.1.2.1 Five Inputs Circuit Experiments. All new gates introduced to the circuit are descendants of the bottom most circuit, causing the graphs in Figure A.7 to become very tall and narrow. The results shown Figure A.8 are nearly identical to the fixed level experiments, except for the average output ancestral entropy. Here $H(P)_{max}$ is 1 because all gates introduced have ancestors from the bottommost and topmost circuits. There are 987 out of 1023 nodes with ancestors from two sub-circuits, and no nodes with ancestors from all three. As Figure A.7(b) shows, the upper rightmost circuit contributes ancestry information from only one of the input nodes. Ancestral entropy values of .85 or higher occur in 986 nodes which is 53.8% of $H(P)_{max}$. Again, much of the original circuit has not been replaced. The original circuit has 18 nodes (all with $H(P)_{min}$ value) while the 1000th iteration has 15 nodes that have a $H(P)_{min}$ value.

4.1.2.2 Five 'Split' Inputs Circuit Experiments. In this configuration, all new gates introduced are descendants of the bottommost circuit. Figure A.9

illustrates the introduction of new nodes to the bottom two levels of the circuits. A total of 15 of 18 original gates remain in the circuit. All new nodes introduced have an ancestral entropy of $H(P)_{min}$. The average ancestral entropy for every level, node and output in the circuit is also $H(P)_{min}$ because all new nodes are descendants from only one original sub-circuit. The two rightmost sub-circuits in iteration zero are preserved. Figure A.10 illustrates these results.

4.1.2.3 Eleven Inputs Experiment. This experiments results are the same as the five ‘split’ input experiment. Levels zero and one both belong to the bottommost circuit, resulting in new nodes having only one circuit ancestor. Again, 15 of the 18 original nodes remain in the circuit. The two leftmost circuits from iteration zero are preserved. These results are illustrated in Figures A.11 and A.12.

4.1.3 LargestLevelTwoGates. The largest level algorithm chooses the first gate from the hierarchical level with the highest number of gates, and the second gate from the same level or the level above. This selection results in shorter and wider circuits.

4.1.3.1 Five Inputs Circuit Experiments. In this experiment, most of the gates are introduced in the middle of the circuit. Figure A.13 shows an increase in width of the graph, even in the early iterations. This circuit has 71 levels, with an average ancestral circuit level entropy of .95. There are 39 levels with the ancestral entropy of 1 or greater. As Figure A.14 shows, the outputs are not replaced so they maintain the value of $H(P)_{min}$. This circuit is much more balanced than the previous circuits. There are 736 out of 1017 nodes with all three sub-circuit ancestors and 13 of 18 original gates remaining in the circuit. There are 466 nodes that are at least 99% of $H(P)_{max}$ while 731 nodes are at least 90% of $H(P)_{max}$. This is expected because in early iterations the largest levels contains nodes from all of the defined sub-circuits, making it more likely for new nodes to have ancestors from all sub-circuits.

4.1.3.2 Five ‘Split’ Inputs Circuit Experiments. In this circuit the output ancestral entropy maintains the original value because outputs are never replaced. The maximum entropy value for experiment is 1 because the right uppermost circuit in Figure A.15(a) is preserved. Two ancestors are in 776 of 1017 nodes, while no nodes have ancestor from all three original sub-circuits. The circuit contains 13 of 18 original nodes.

4.1.3.3 Eleven Inputs Experiment. This circuit has 72 levels with an average circuit level ancestral entropy of .44. Ten of the levels have entropy values of at least .85, and 17 are at least .8, 679 of 1022 nodes have ancestral entropy values between .9 and 1, inclusive, 673 nodes have ancestral entropy between .95 and 1, while 446 nodes are greater or equal to .99. One ancestor from an original sub-circuit is in 323 nodes, 699 have two ancestors, no gates have ancestors from all three original sub-circuits, and 15 of 18 original circuits remain. Figures A.17 and A.18 illustrate this.

4.1.4 RandomLevelTwoGates. The random level algorithm selects the first gate from the circuit in a random, uniform manner. The second gate is selected either from same level, the level above, or the level below the first selected gate. Although this algorithm is more random than the aforementioned algorithms and is likely to produce better component hiding properties, it limits the number of possible replacement sub-circuits because both gates selected for replacement are from adjacent or equivalent levels.

4.1.4.1 Five Inputs Circuit Experiments. The resulting circuit has 129 levels; 70 are above the $H(P)_{min}$ value of zero with respective ancestral averages of .39 and .73. At the 500th iteration the average output ancestral entropy is 1.164. The circuit contains only 2 of 23 original gates. There are a total of 511 nodes in the circuit and 340 have one circuit ancestor, 108 have two ancestors, and 63 have three

ancestors. Only seven nodes are 99% of $H(P)_{max}$, and 51 nodes are 90% of $H(P)_{max}$ (1.585). Figures A.19 and A.20 illustrate this.

4.1.4.2 Five ‘Split’ Inputs Circuit Experiments. This configuration introduces a sub-graph at 6th iteration and removes this sub-graph from the graph at the 79th iteration. This is not an original circuit being removed. There are the same 129 levels, since this is the same circuit as the five input experiment, with 113 above the $H(P)_{min}$ value of zero. The total average ancestral entropy average is .64 and the average for nodes with values above zero is .73, the same as in the five input circuit. At the 500th iteration the average output ancestral entropy is .996. There are a total of 511 nodes in the circuit and 197 have one original circuit ancestor, 257 have two ancestors, and 57 have ancestors from three original sub-circuits. Only two of the original 18 gates remain in the circuit as Figures A.21 and A.22 show.

4.1.4.3 Eleven Inputs Experiment. There are 207 levels with 64 levels at $H(P)_{min}$ and 92 levels greater than zero or equal to one in this circuit. 52 nodes have entropy values greater than one. The average ancestral entropy for each level in the circuit is .66. The average for the nodes greater than zero is .95, meaning that on average every node has ancestors from two out of three circuits. At the 500th iteration the average output ancestral entropy is 1.56 which is 98.4% $H(P)_{max}$. At the 1000th iteration it is 1.468, which is 92.6% of $H(P)_{max}$. All original gates in the circuit have been replaced at least once. Figures A.23 and A.24 illustrates this.

4.1.5 RandomTwoGates. The random two gate selection algorithm chooses the first gate in a random, uniform manner. The second gate is chosen in the same way. This algorithm is truly the most random and is expected to produce the best component hiding properties for each of the circuit variants.

4.1.5.1 Five Inputs Circuit Experiments. This circuit, because of how the ancestry file is built with the all inputs connected to one sub-circuit, has two

of the original circuits replaced with new gates at iteration 66 and 72. There are 449 levels with an average ancestral entropy of .169. There are only 91 levels with ancestral entropy values greater than zero because the upper rightmost circuit in Figure A.25(a) dominates the graph as both of the other original circuits get removed from the graph in early iterations. Figure A.25(e) shows only one remaining node from the original bottommost circuit in Figure A.25(a). At the 1000th iteration the average ancestral entropy at the outputs is 1.516, 882 gates have only one ancestor from an original circuit, 63 have two ancestors, and 74 have three. All original gates are replaced at least once in this circuit.

4.1.5.2 Five ‘Split’ Inputs Circuit Experiments. As in the five input circuit, there are 449 levels with an average ancestral entropy per level of 1.126. There are 417 levels that have a greater than zero entropy value. Of these, the average is 1.215. There are 285 levels that have an entropy value greater than one and their average is 1.446. At the 1000th iteration the output ancestral entropy average is 1.559. At the 710th iteration this average is 1.539. There are 177 gates with ancestors from only one of the original circuits, 244 with ancestors from two and 598 gates with ancestors from three. There are 150 gates with an ancestral entropy value of 99% of $H(P)_{max}$ and there are 569 gates with an ancestral entropy value of 90% $H(P)_{max}$. All original gates have been replaced at least once. Figures A.27 and A.28 illustrate this.

4.1.5.3 Eleven Inputs Experiment. In this circuit there are 401 levels with an average ancestral entropy per level of .858, 345 of the levels are greater than zero, while 152 levels are at least one. The average output ancestral entropy at the 1000th iteration is 1.516 and first reaches this value at the 561st iteration. There are 257 gates with ancestors from only one original circuit, 463 with ancestors from two circuits, and 304 with ancestors from three circuits. There are 15, 191, and 270 nodes with ancestral entropy values of 99%, 95%, and 90% of $H(P)_{max}$ respectively. All original gates have been replaced at least once as Figures A.29 and A.30 show.

4.1.6 RandomAlgorithm. The random algorithm chooses one of the previously described algorithms at random and performs its function. The algorithms can be weighted programmatically but are equally weighted herein. This algorithm is not expected to perform as well as the random two gate selection because all of the algorithms, except for random two gate, are deterministic in nature. Furthermore, because the fixed level in the FixedLevelTwoGates algorithm is set to one, it behaves similarly to the OutputLevelTwoGates algorithm.

4.1.6.1 Five Inputs Circuit Experiments. In this circuit two of the three original circuits are removed at iteration 113 and 176. There are 355 levels in this circuit. The average ancestral entropy for all levels is 1.136. 102 of the levels are $H(P)_{max}$, while 227 are at least equal to one (two ancestors). The output ancestral entropy is 1.58 (nearly $H(P)_{max}$) and reaches this value at the 80th iteration. There are 254 gates with one ancestor, 225 with two ancestors, and 522 with three ancestors. There are 405 nodes that are at least 99% of $H(P)_{max}$, 440 that are at least 95% of $H(P)_{max}$, and 492 that are at least 90% of $H(P)_{max}$. The average node ancestral entropy is 1.023. All original gates in the circuit are replaced at least once. Figures A.31 and A.32 illustrate this.

4.1.6.2 Five ‘Split’ Inputs Circuit Experiments. The five ‘split’ input circuit has 355 levels with 102 of them equivalent to $H(P)_{max}$, 231 of the levels have an entropy value of at least one. The average output ancestral entropy value at the 1000th iteration is 1.58, and is 1.56 at the 500th iteration. There are 116 gates with one ancestor, 524 with two ancestors, and 467 with three ancestors. There are 152 nodes that are at least 99% of $H(P)_{max}$, 339 that are at least 95% of $H(P)_{max}$, and 429 that are at least 90% of $H(P)_{max}$. All original gates in the circuit have been replaced at least once. Figures A.33 and A.34 illustrate this.

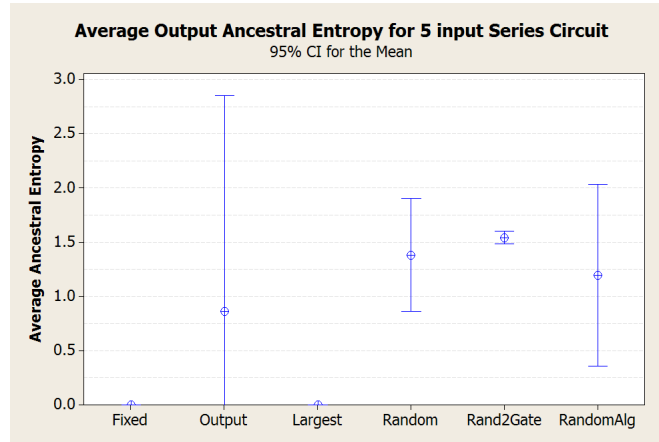
4.1.6.3 Eleven Inputs Experiment. This circuit has 391 levels with an average level ancestral entropy value of 1.264. 374 of the levels are at least zero, and

299 are at least one. The output ancestral entropy reaches 1.58 at the 409th iteration and is $H(P)_{max}$ at the 1000th iteration. There are 131 gates with one circuit ancestor, 194 gates with two ancestors, and 687 gates with three ancestors. All original gates in the circuit have been replaced at least once. Figures A.35 and A.36 illustrate this.

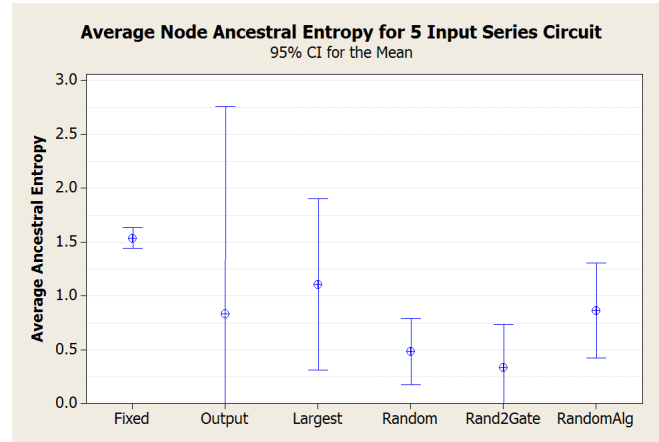
4.1.7 Series Circuits Results Summary. These results illustrate the function performed by each algorithm. Since the component hiding properties are of interest, any further examination of the FixedLevelTwoGates, LargestLevelTwoGates and OutputLevelTwoGates algorithms is excluded. The FixedLevelTwoGates and OutputLevelTwoGates algorithms extrude new gates from the bottom of the circuit while many of the components to hide are left untouched. The OutputLevelTwoGates algorithm can produce the full range of ancestral entropy values at the outputs based on how the sub-circuits are defined, while affecting few or none of the original sub-circuits. The LargestLevelTwoGates algorithm introduces gates to the largest level, while making few changes to other parts of the circuit. Figure 4.3-4.5 show the confidence intervals for the results of each algorithm and circuit configuration.

The results of RandomLevelTwoGates, RandomTwoGates and RandomAlgorithm are similar and don't focus circuit transformations on one part of the circuit. These algorithms replace most or all of the original gates in the circuits at least once, which is necessary to hide circuit components. The confidence intervals for the percentage of original gates remaining in the circuit is illustrated in Figure 4.6. The results of the RandomAlgorithm experiments are not significantly different than the RandomTwoGates and RandomLevelTwoGates results for the five input average output level ancestral entropy, the 11 input average output and circuit level ancestral entropy and for the five 'split' input node and circuit level average ancestral entropy. In the four remaining experiments the results are significantly different than RandomTwoGates and RandomLevelTwoGates and the entropy values are higher.

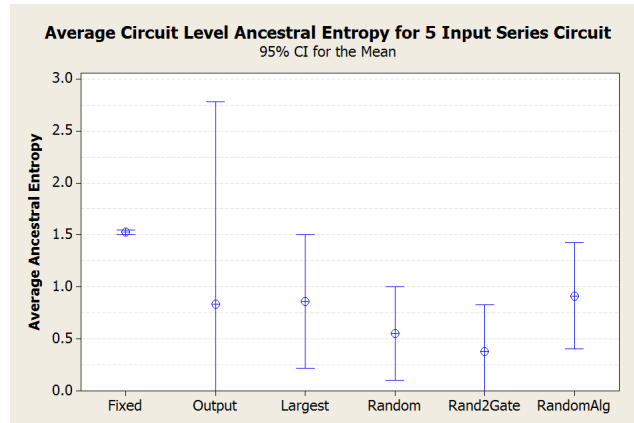
A 2-Sample t test is performed comparing RandomAlgorithm against RandomTwoGates and RandomLevelTwoGates. Resulting boxplots are shown in Figures 4.7-



(a) Output Ancestral Entropy Results

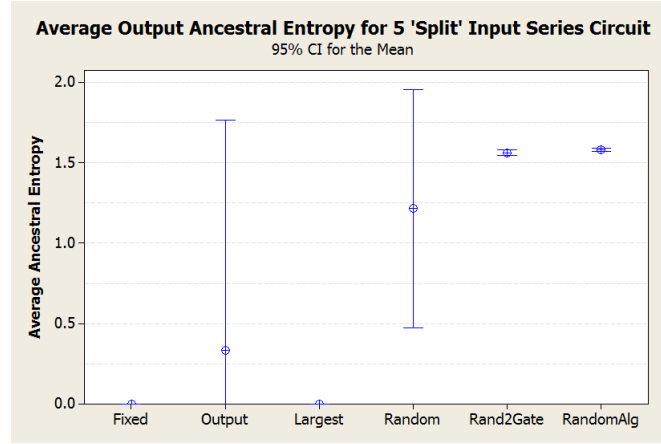


(b) Node Ancestral Entropy Results

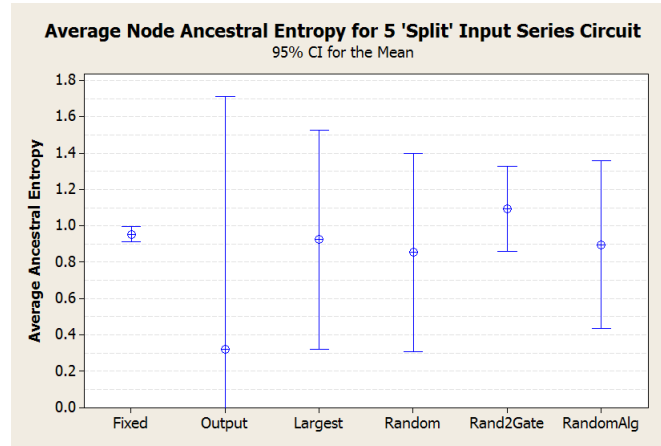


(c) Circuit Level Ancestral Entropy Results

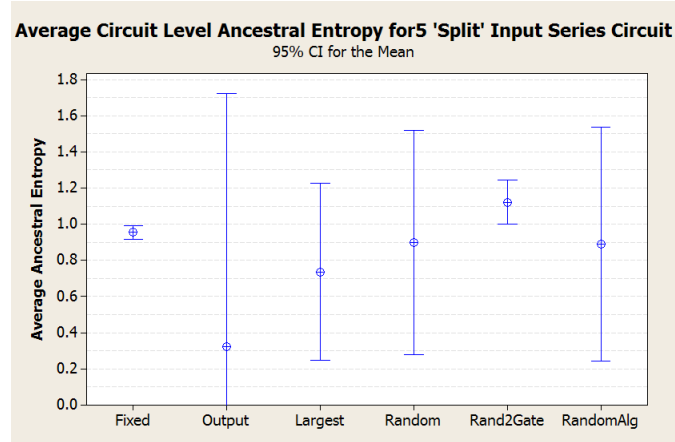
Figure 4.3: Average Ancestral Entropy Intervals for 5 Input Series Circuit (a), (b) and (c)



(a) Output Ancestral Entropy Results

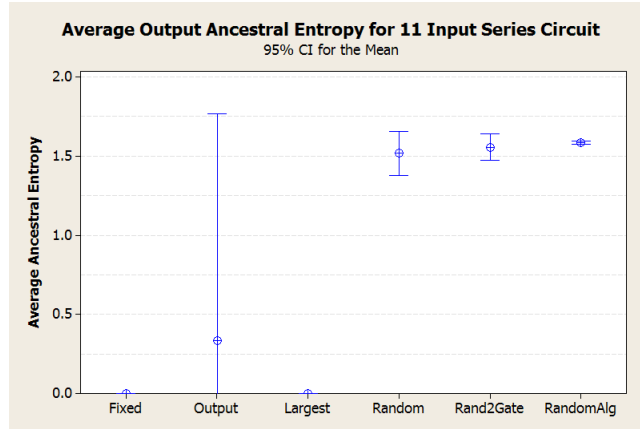


(b) Node Ancestral Entropy Results

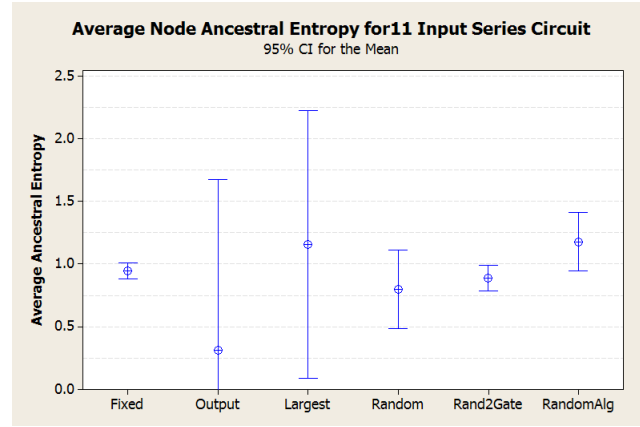


(c) Circuit Level Ancestral Entropy Results

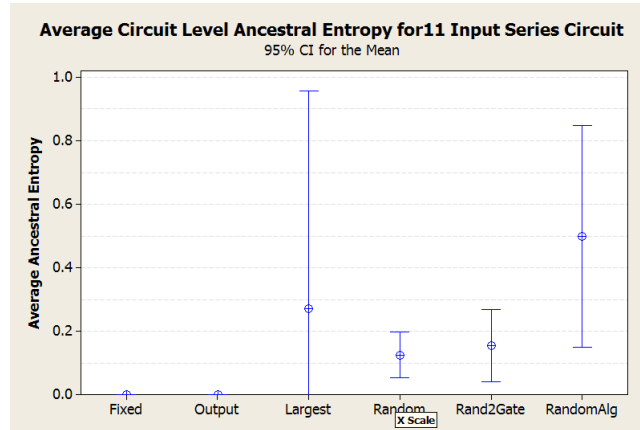
Figure 4.4: Average Ancestral Entropy Intervals for 5 Split Input Series Circuit (a), (b) and (c)



(a) Output Ancestral Entropy Results

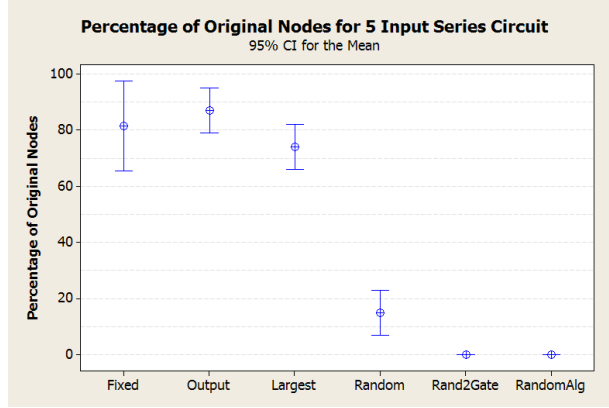


(b) Node Ancestral Entropy Results

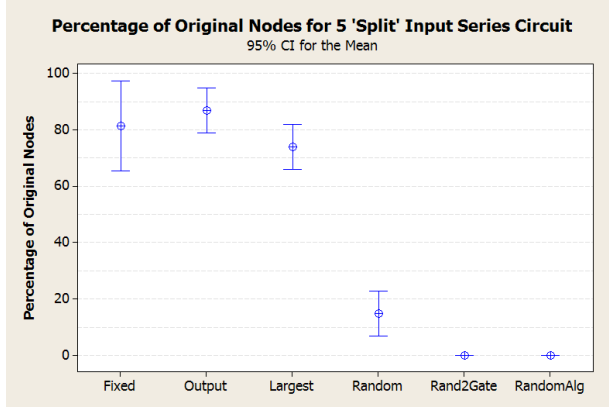


(c) Circuit Level Ancestral Entropy Results

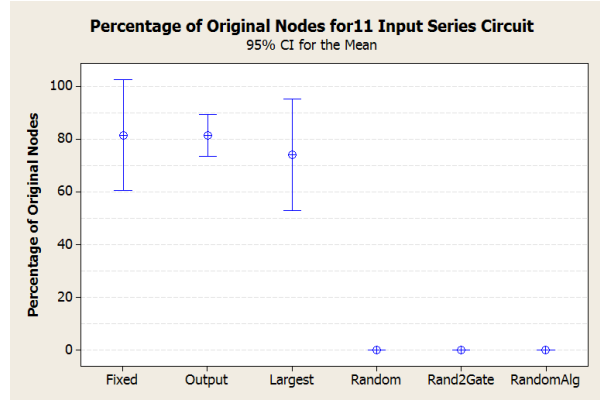
Figure 4.5: Average Ancestral Entropy Intervals for 11 Input Series Circuit (a), (b) and (c)



(a) Percentage of Original gates in 5 Input Circuits

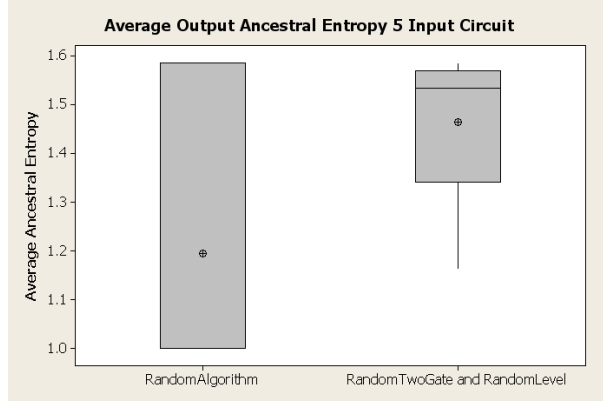


(b) Percentage of Original gates in 5 Split Input Circuits

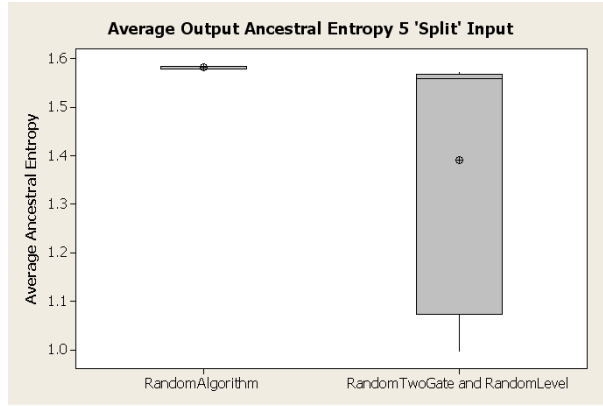


(c) Percentage of Original gates in 11 Input Circuits

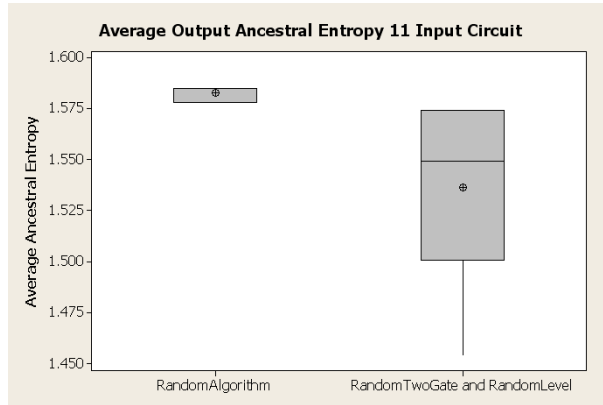
Figure 4.6: Percentage of Original Gates in Series Circuits (a), (b) and (c)



(a) Boxplot for 5 Input Circuits

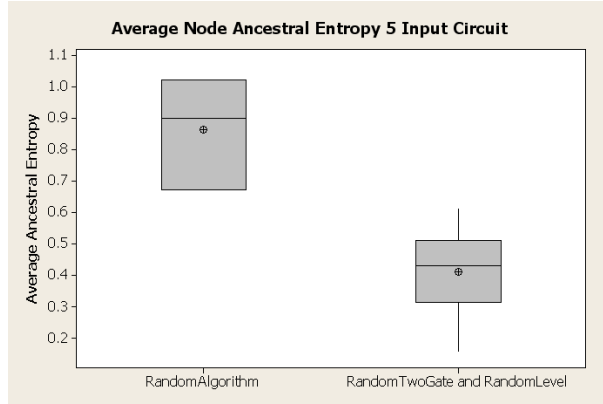


(b) Boxplot for 5 Split Input Circuits

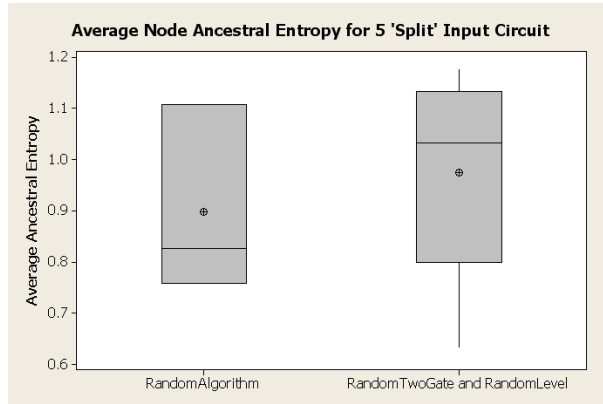


(c) Boxplot for 11 Input Circuits

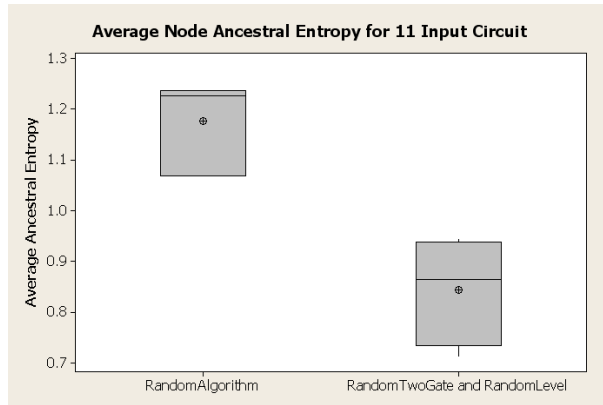
Figure 4.7: Boxplot from 2-Sample t Test of Average Output Ancestral Entropy with Random Algorithms on Series Circuits (a), (b) and (c)



(a) Boxplot for 5 Input Circuits

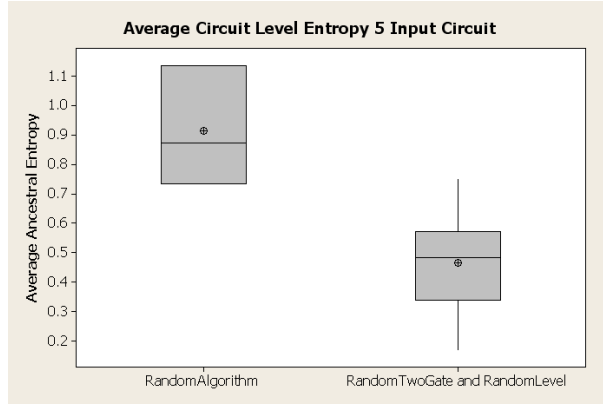


(b) Boxplot for 5 Split Input Circuits

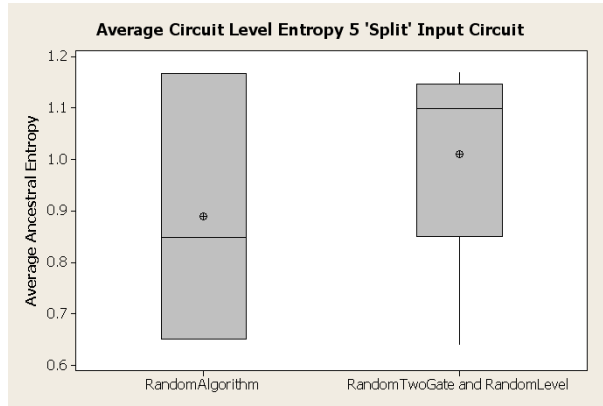


(c) Boxplot for 11 Input Circuits

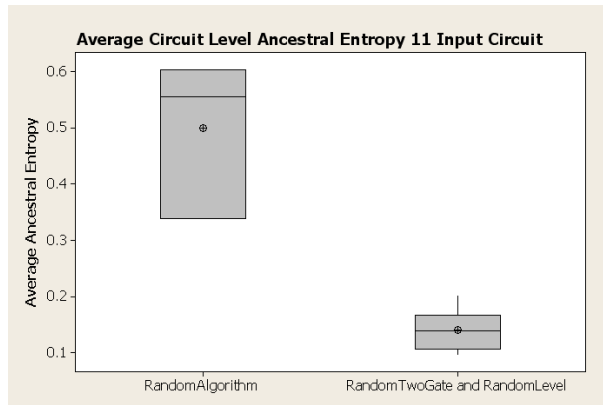
Figure 4.8: Boxplot from 2-Sample t Test of Average Node Ancestral Entropy with Random Algorithms on Series Circuits (a), (b) and (c)



(a) Boxplot for 5 Input Circuits



(b) Boxplot for 5 Split Input Circuits



(c) Boxplot for 11 Input Circuits

Figure 4.9: Boxplot from 2-Sample t Test of Average Level Ancestral Entropy with Random Algorithms on Series Circuits (a), (b) and (c)

Table 4.2: P-Values from 2-Sample t Test comparing RandomAlgorithm against RandomTwoGates and RandomLevelTwoGates Using Series Circuits

| Entropy | 5 Input | 5 ‘Split’ Input | 11 Input |
|---------------|-------------|-----------------|--------------|
| Output Level | .137 | .273 | .136 |
| Node | .005 | .595 | .002 |
| Circuit Level | .013 | .459 | 0.000 |

4.9. Table 4.2 contains the p values from the 2-Sample t test. RandomAlgorithm performs significantly better than RandomTwoGates and RandomLevelTwoGates in terms of node and circuit level ancestral entropy for the 5 and 11 Input experiments.

4.2 Three Gate Replacement C-17 Parallel Circuits

The parallel C-17 circuits show how independent or geographically separated circuits are hidden by taking advantage of circuit merging during the creation of the circuit variants. Figure 4.10 shows the high level parallel circuit configurations. The algorithms perform the same function as described earlier.

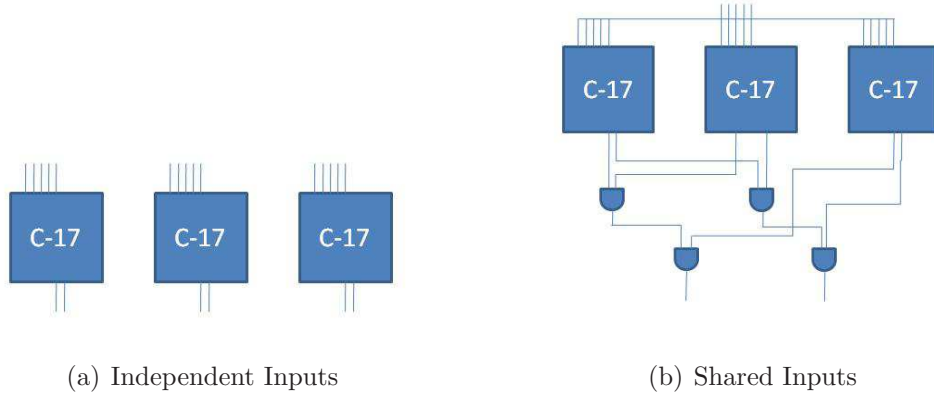


Figure 4.10: C-17 Parallel Circuits
(a) and (b)

The shared input circuit in Figure 4.10(b) is composed of three C-17 sub-circuits, all using the same five inputs. The outputs of each sub-circuit are routed through a series of *AND* gates providing two circuit outputs.

The parallel circuits are three independent C-17 circuits with independent inputs. This configuration is chosen because these circuits represent like circuits that are unrelated geographically separated circuits. Figure 4.10(a) illustrates this circuit configuration.

4.2.1 *FixedLevelRandomTwoGates.*

4.2.1.1 *Shared Inputs.* The final circuit variant has 253 levels. The average ancestral entropy for each level is 1.86. Three levels have an average ancestral entropy value of zero (all original gates), 11 levels have an average number of ancestors from one original sub-circuit, 10 from two sub-circuits, and 229 levels from three sub-circuits. The average output ancestral entropy is zero because the fixed level is set to one, so the outputs are never selected for replacement. There are 33 nodes in the circuit with only one ancestor from an original sub-circuit, 44 with two ancestors, 69 with three ancestors, and 881 with four ancestors. This information may be misleading because much of the original circuit is retained, but all new gates with four ancestors are extruded from the bottom of the circuit. The average node ancestral entropy is 1.99, while the $H(P)_{max}$ value for this *four* component circuit equals 2, 830 nodes have an ancestral entropy of at least 99% of $H(P)_{max}$, while 881 nodes have an ancestral entropy of at least 95% of $H(P)_{max}$. The circuit contains 16 of the 22 original gates. These results are illustrated in Figure B.6.

4.2.1.2 *Individual Inputs.* In this circuit there are 143 levels. The average ancestral entropy for each level is 1.376. Six levels have an entropy value of zero (one ancestor), 14 levels have an average ancestral entropy value between zero and one (ancestry values from two sub-circuits), while 124 have an entropy value greater than one (ancestry values from three sub-circuits). There are 89 nodes with only one sub-circuit ancestor, 45 nodes with two ancestors and 899 nodes with three ancestors. 839 nodes have an ancestral entropy of at least 99% of $H(P)_{max}$, while 892 nodes have an ancestral entropy of at least 95% of $H(P)_{max}$ and 898 nodes have

an ancestral entropy of at least 90% of $H(P)_{max}$. The circuit contains 11 of the 18 original gates. These results are illustrated in Figures B.3 and B.4.

4.2.2 *OutputLevelRandomTwoGates.*

4.2.2.1 *Shared Inputs.* There are 143 levels in this circuit. The average level ancestral entropy for all levels is 1.525, while $H(P)_{max} = 2$ for this circuit. Four levels have an entropy value of zero, 18 have an entropy value of one, and 451 have an entropy value between one and 1.585 which is the $H(P)_{max}$ value for a 3 component circuit. The output ancestral entropy average is 1.585 at the 1000th iteration, and reaches it at the 868th iteration. There are 29 gates in the circuit with only one sub-circuit ancestor, 31 with two ancestors and 967 with three ancestors. There are no nodes with all four ancestors. Because the gates selected for replacement come from the bottom two levels of the graph, 18 of the 22 original gates remain in the circuit. These results are illustrated in Figures B.5 and B.6.

4.2.2.2 *Individual Inputs.* This circuit has 186 levels with an average ancestral entropy for all levels of 1.42. There are 7 levels with an ancestral entropy value equal to zero, 11 levels with a value of one, and 168 levels with an entropy value greater than one. The average output ancestral entropy reaches $H(P)_{max}$ at the 640th iteration. There are 72 nodes with one sub-circuit ancestor, 43 nodes with two ancestors and 918 nodes with three ancestors, 863 nodes have an ancestral entropy of at least 99% of $H(P)_{max}$, while 917 nodes have an ancestral entropy of at least 95% of $H(P)_{max}$ and 918 nodes have an ancestral entropy of at least 90% of $H(P)_{max}$. The circuit contains 12 of the 18 original gates. These results are illustrated in Figures B.7 and B.8.

4.2.3 *LargestLevelRandomTwoGates.*

4.2.3.1 *Shared Inputs.* This circuit has 63 levels with an average ancestral entropy for all levels of .814. There are 7 levels with an ancestral entropy

value of zero, 27 levels with a value between zero and one, and 29 levels greater than one. The output ancestral entropy equals zero because the output gates are never replaced. There are 220 nodes with only one sub-circuit ancestor, 338 nodes with two ancestors, and 465 nodes with three ancestors. Although $H(P)_{max}$ for this circuit is 2, the maximum entropy value that can be reached is 1.585 because there are no nodes in this circuit containing ancestors from all original sub-circuits. 15 of 22 original gates remain in the circuit. These results are illustrated in Figures B.9 and B.10.

4.2.3.2 Individual Inputs. There are 64 levels in this circuit with an average ancestral entropy for all levels of 1.025. There are four levels with an ancestral entropy value of zero, 25 levels with a value between zero and one, and 35 levels with a value greater than one. Again, the output ancestral entropy equals zero because the output gates are never replaced, 404 nodes have an ancestral entropy of at least 99% of $H(P)_{max}$, while 620 nodes have an ancestral entropy of at least 95% of $H(P)_{max}$ and 624 nodes have an ancestral entropy of at least 90% of $H(P)_{max}$; 10 of 18 original gates remain in the circuits. These results are illustrated in Figures B.11 and B.12.

4.2.4 RandomLevelTwoGates.

4.2.4.1 Shared Inputs. There are 227 levels in this circuit with an average ancestral entropy for all levels of .989. There are 17 levels with an ancestral entropy value of zero (only one sub-circuit ancestor), 111 levels with a value between zero and one (two sub-circuit ancestors), 59 levels with a value between 1 and 1.585 (three sub-circuit ancestors), and 40 levels with a value greater than 1.585 (four sub-circuit ancestors). The average output ancestral entropy is 1.96 after the 513th iteration (98% of $H(P)_{max}$). There are 293 nodes with one sub-circuit ancestor, 336 nodes with two ancestors, 221 nodes with three ancestors and 153 nodes with four ancestors. The average node ancestral entropy is .96; 63 nodes have an ancestral entropy of at least 99% of $H(P)_{max}$, while 142 nodes have an ancestral entropy of at least 95% of $H(P)_{max}$ and 153 nodes have an ancestral entropy of at least 90% of

$H(P)_{max}$ (all nodes with four ancestors have generally evenly balanced ancestry values). Only 1 of 22 original gates remains in the circuits. These results are illustrated in Figures B.13 and B.14.

4.2.4.2 Individual Inputs. There are 188 levels with an average ancestral entropy for all levels of .73. There are 13 levels with an ancestral entropy value of zero, 139 level with a value between zero and one, and 36 levels with a value greater than one. The average output ancestral entropy is .9 after iteration 811 (56.7% of $H(P)_{max}$). There are 333 nodes with one ancestor, 430 nodes with two ancestors and 253 nodes with three ancestors. The average node ancestral entropy is .77. There are 73 nodes with an ancestral entropy of at least 99% of $H(P)_{max}$, while 176 nodes have an ancestral entropy of at least 95% of $H(P)_{max}$ and 232 nodes have an ancestral entropy of at least 90% of $H(P)_{max}$. Only 1 of 18 original gates remains in the circuit. These results are illustrated in Figures B.15 and B.16.

4.2.5 RandomTwoGates.

4.2.5.1 Shared Inputs. There are 355 levels with an average ancestral entropy for all levels of .77. There are 56 levels with an ancestral entropy value of zero, 202 levels with a value between zero and one, 90 levels with a value between 1 and 1.585 and 7 with a value greater than 1.585. The average output ancestral entropy is 1.774 after the 875th iteration (88.5% of $H(P)_{max}$). There are 384 nodes with one sub-circuit ancestor, 407 nodes with two ancestors, 222 nodes with three ancestors and 11 nodes with four ancestors. The average node ancestral entropy is .725. No nodes have an ancestral entropy of at least 99% of $H(P)_{max}$, while 11 nodes have an ancestral entropy of at least 95% of $H(P)_{max}$. All original gates in the circuit have been replaced at least once. These results are illustrated in Figures B.17 and B.18.

4.2.5.2 Individual Inputs. There are 344 levels with an average ancestral entropy for all levels of .822. There are 25 levels with an ancestral entropy value of zero, 218 levels with a value between zero and one and 101 levels with a value greater than one. The average output ancestral entropy is 1.34 after the 991st iteration (84.5% of $H(P)_{max}$). There are 321 nodes with one sub-circuit ancestor, 394 nodes with two ancestors and 313 nodes with three ancestors. The average node ancestral entropy is .81. There are 66 nodes with an ancestral entropy of at least 99% of $H(P)_{max}$, while 214 nodes have an ancestral entropy of at least 95% of $H(P)_{max}$ and 263 nodes have an ancestral entropy of at least 90% of $H(P)_{max}$. All original gates have been replaced at least once. These results are illustrated in Figures B.19 and B.20.

4.2.6 RandomAlgorithm.

4.2.6.1 Shared Inputs. In this circuit there are 282 levels with an average ancestral entropy for all levels of 1.21. There are 42 levels with an ancestral entropy value of zero, 74 levels with a value between zero and one, 39 levels with a value between 1 and 1.585 and 127 with a value greater than 1.585. The average output ancestral entropy is 1.99 after iteration 875 (99.5% of $H(P)_{max}$). There are 324 nodes with one sub-circuit ancestor, 141 nodes with two ancestors, 270 nodes with three ancestors and 271 nodes with four ancestors. The average node ancestral entropy is 1.05. 108 nodes have an ancestral entropy of at least 99% of $H(P)_{max}$, while 258 nodes have an ancestral entropy of at least 95% of $H(P)_{max}$ and 271 nodes have an ancestral entropy at least 90% of $H(P)_{max}$. All original gates in the circuit have been replaced at least once. These results are illustrated in Figures B.21 and B.22.

4.2.6.2 Individual Inputs. This circuit has 251 levels with an average ancestral entropy for all levels of .905. There are 32 levels with an ancestral entropy value of zero, 106 levels with a value between zero and one, and 113 levels with a

value greater than one. The average output ancestral entropy is 1.58 after iteration 991 (99.7% of $H(P)_{max}$). There are 259 nodes with one ancestor, 243 nodes with two ancestors and 528 nodes with three ancestors. The average node ancestral entropy is .997. 219 nodes have an ancestral entropy of at least 99% of $H(P)_{max}$, while 391 nodes have an ancestral entropy of at least 95% of $H(P)_{max}$ and 451 nodes have an ancestral entropy of at least 90% of $H(P)_{max}$. All original gates in the circuit have been replaced at least once. These results are illustrated in Figures B.23 and B.24.

4.2.7 Parallel Circuits Results Summary. These parallel circuit results illustrate the functions performed by each algorithm. As with the series circuits the component hiding properties are of greatest interest and any further examination of the FixedLevelTwoGates, LargestLevelTwoGates and OutputLevelTwoGates algorithms is excluded. Figure 4.11-4.12 illustrate the confidence intervals for the result of each algorithm and circuit configuration.

These results are similar to the results in the series experiments. RandomLevelTwoGates, RandomTwoGates and RandomAlgorithm, provide similar results and don't focus circuit transformations only one part of the circuit. In all three experiments, RandomLevelTwoGates replaces all original nodes but one, while RandomTwoGates and RandomAlgorithm replace all original nodes in the circuit. This is a necessary attribute if a circuit is to have any component hiding properties. The confidence intervals for the percentage of original gates remaining in the circuit is illustrated in Figure 4.13.

A 2-Sample t test is performed comparing RandomAlgorithm to RandomLevelTwoGates and RandomTwoGates. The resulting boxplots are shown in Figures 4.14-4.16. Table 4.3 contains the p values from the 2-Sample t test. RandomAlgorithm is significantly different than RandomLevelTwoGates and RandomTwoGates in terms of circuit level entropy using the shared input circuit and in terms of output level entropy using the individual input circuit. The shared input node entropy p value of

Table 4.3: P-Values from 2-Sample t Test comparing RandomAlgorithm against RandomTwoGates and RandomLevelTwoGates Using Parallel Circuits

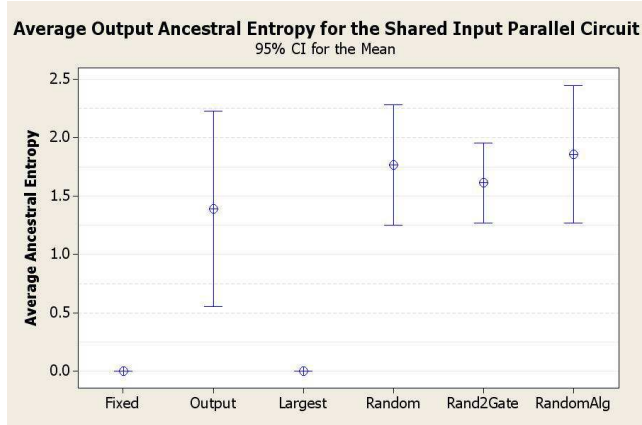
| Entropy | Shared Input | Individual Input |
|---------------|--------------|------------------|
| Output Level | .267 | .002 |
| Node | .060 | .079 |
| Circuit Level | .019 | .500 |

.060 is close to being different than the others, and more experiments are needed to determine if this is better.

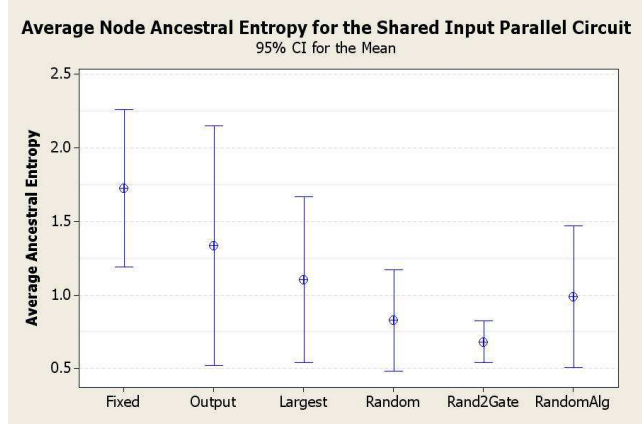
4.3 Three and Four Gate Replacement Circuit Reduction

Allowing redundant and symmetric gates as replacements causes many buffers or constant output gates to be inserted into the circuit. The initial assumption is the insertion of these buffers allows better circuit transformations in future experiment iterations. However, this is not the case. In [9], Kim shows using a circuit reduction algorithm and three gate replacement circuits that allow redundant and symmetric gates can be nearly reduced back to the original circuit configuration. Figure 4.17, shows a circuit variant after 1000 iterations using the RandomAlgorithm algorithm. The original circuit contains 18 gates and the variant contains 1096 gates. Figure 4.18, illustrates the circuit variant after being reduced. This circuit contains only 123 gates. His results show performing fewer iterations to create the circuit variant allows circuits to be reduced even closer to the original.

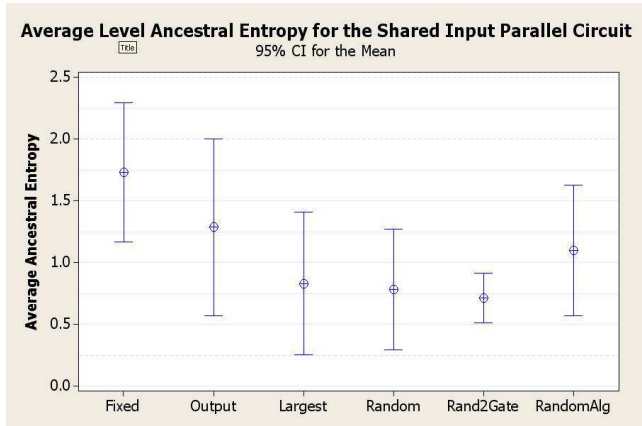
However, reducing circuit variants created with a two gate selection and four gate replacement were unsuccessful. Figure 4.19 shows a circuit variant at the 1000th iteration. Figure 4.20 illustrates the reduced circuit. The gate count in the original is 1096 gates, while the reduced version has 496. When using a two gate selection and a four gate replacement it is not necessary to allow redundant or symmetric gates to cause independent circuits to merge, resulting in a circuit that is not as easily reduced. The next section examines the entropy properties of the algorithms when performing four gate replacements.



(a) Output Ancestral Entropy Results

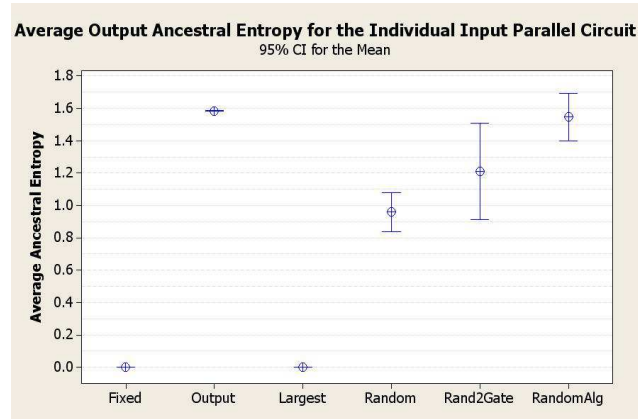


(b) Node Ancestral Entropy Results

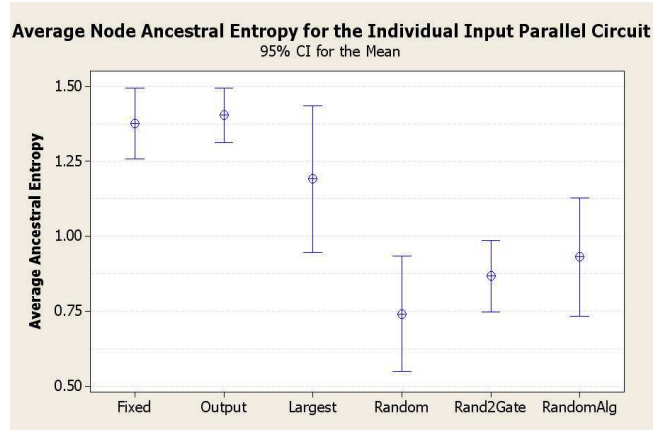


(c) Circuit Level Ancestral Entropy Results

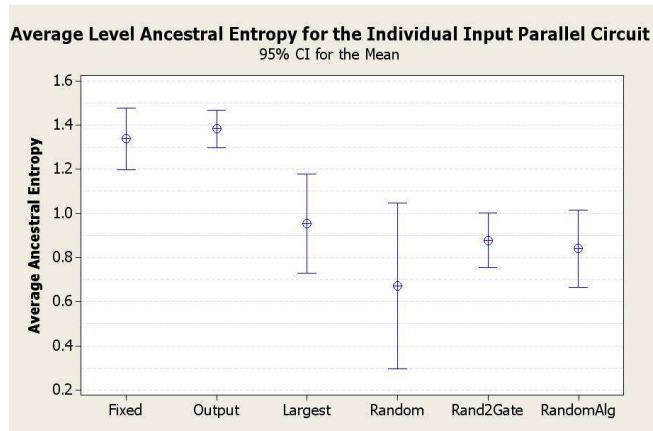
Figure 4.11: Average Ancestral Entropy Intervals for Shared Input Parallel Circuit (a), (b) and (c)



(a) Output Ancestral Entropy Results

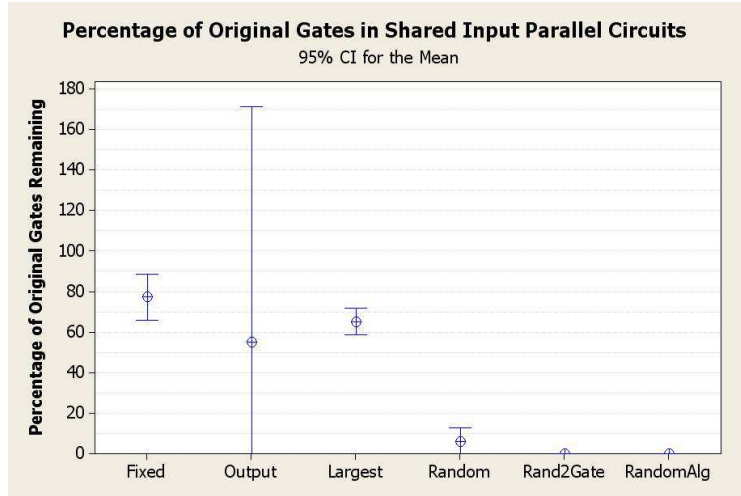


(b) Node Ancestral Entropy Results

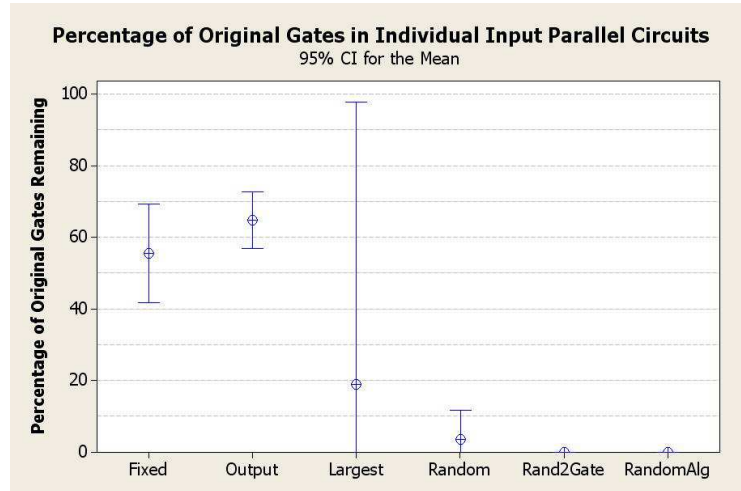


(c) Circuit Level Ancestral Entropy Results

Figure 4.12: Average Ancestral Entropy Intervals for Individual Input Parallel Circuit (a), (b) and (c)

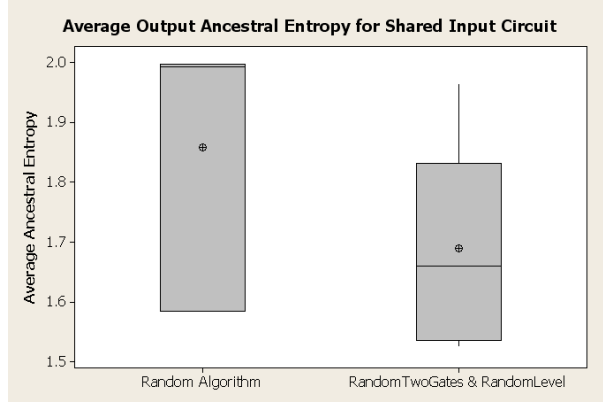


(a) Percentage of Original gates in Shared Input Circuits

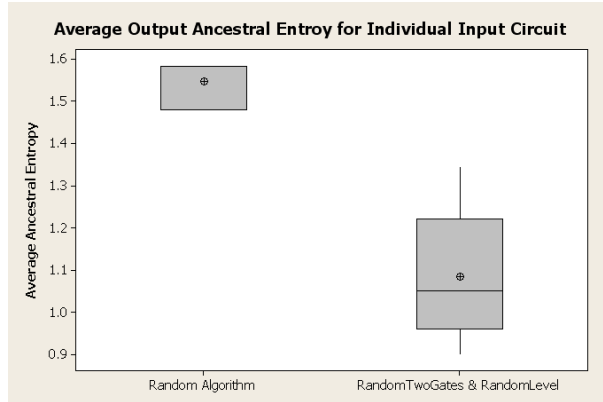


(b) Percentage of Original gates in Individual Input Circuits

Figure 4.13: Percentage of Original Gates in Parallel Circuits (a) and (b)

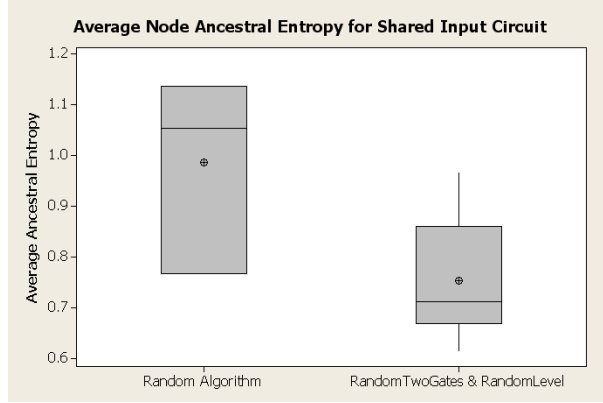


(a) Boxplot for Shared Input Parallel Circuits

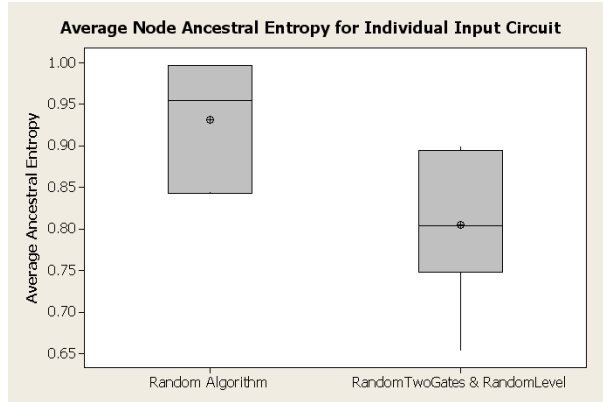


(b) Boxplot for Individual Input Parallel Circuits

Figure 4.14: Boxplot from 2-Sample t Test of Average Output Ancestral Entropy with Random Algorithms on Parallel Circuits (a) and (b)

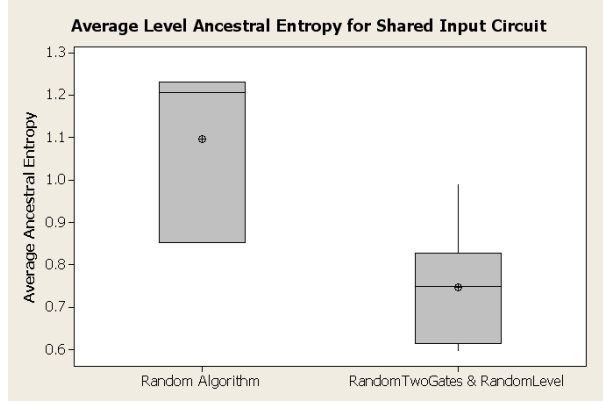


(a) Boxplot for Shared Input Parallel Circuits

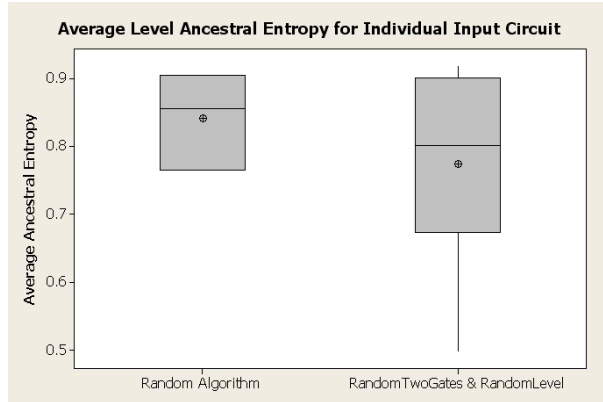


(b) Boxplot for Individual Input Parallel Circuits

Figure 4.15: Boxplot from 2-Sample t Test of Average Output Ancestral Entropy with Random Algorithms on Parallel Circuits (a) and (b)



(a) Boxplot for Shared Input Parallel Circuits



(b) Boxplot for Individual Input Parallel Circuits

Figure 4.16: Boxplot from 2-Sample t Test of Average Output Ancestral Entropy with Random Algorithms on Parallel Circuits (a) and (b)

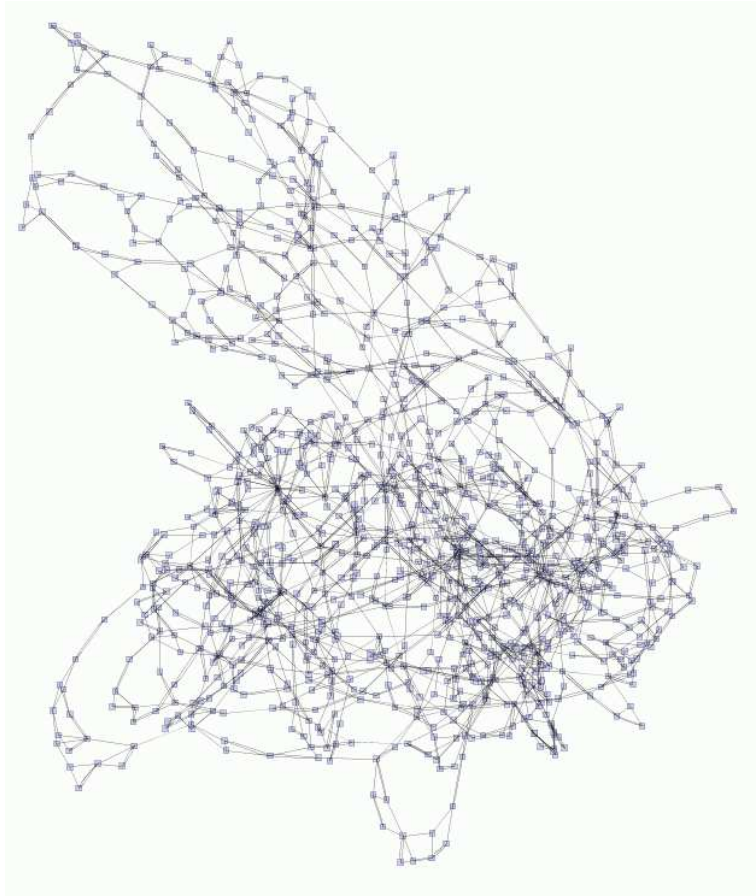


Figure 4.17: 3 Gate Replacement C-17 Circuit Variant at 1000 iterations

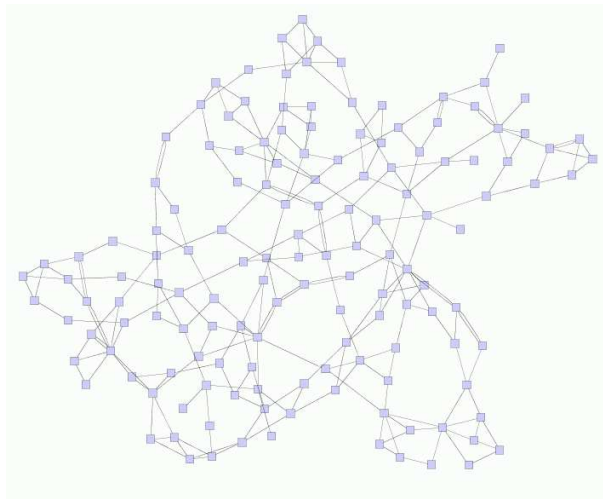


Figure 4.18: 3 Gate Replacement C-17 Circuit Reduced

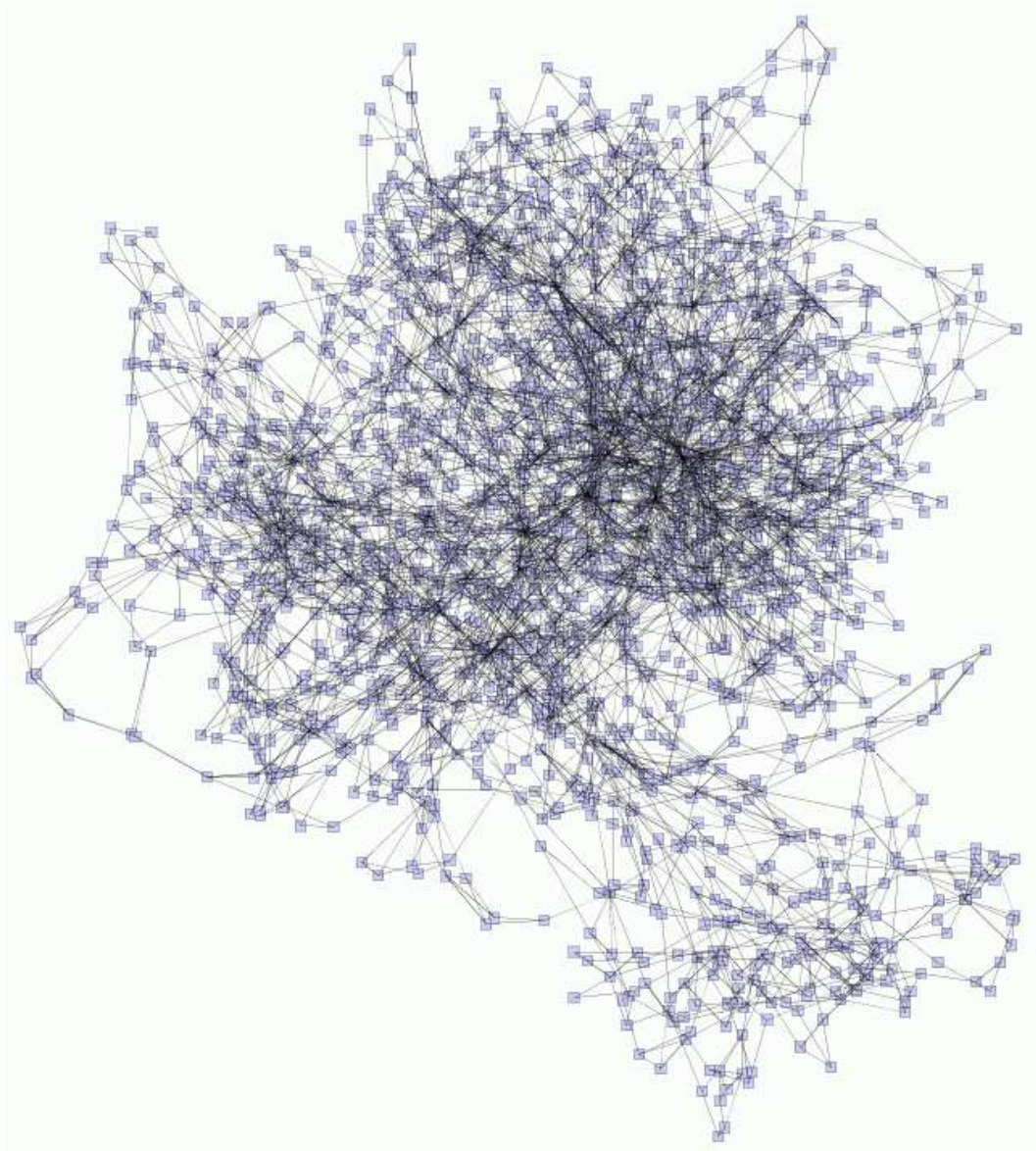


Figure 4.19: 4 Gate Replacement C-17 Circuit Variant at 1000 iterations

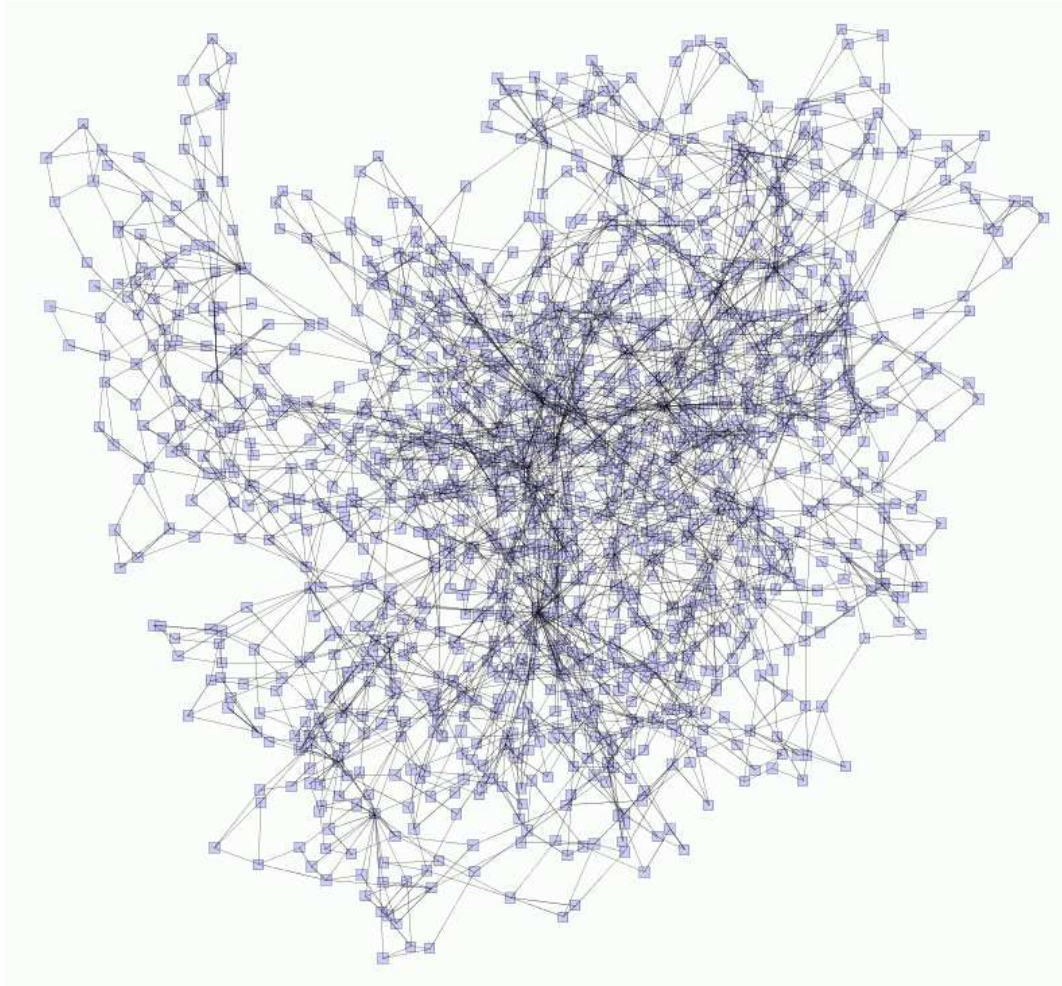


Figure 4.20: 4 Gate Replacement C-17 Circuit Reduced

4.4 Four Gate Replacement C-17 Series Circuits

The two gate selection with a four gate replacement experiments produce similar entropy results as the three gate replacement experiments. This section illustrates the confidence intervals and the boxplots for series and parallel four gate replacement circuits. The series circuits used here are the same circuits used in the three gate replacement experiments. Four gate replacements are evaluated redundant and dual input gates are not allowed to be introduced into the circuit. Since the FixedLevelTwoGate, OutputLevelTwoGate and the LargestLevelTwoGate algorithms don't remove many of the of original gates of a circuit, they are not discussed here.

Comparing the four gate replacement results in Appendix C with the three gate replacements in Appendix A, the four gate replacement experiments generally have higher ancestral entropy results for the RandomLevelTwoGates, RandomTwoGates and RandomAlgorithm experiments. These results are displayed in Tables 4.4-4.6.

The average ancestral entropy intervals and percentage for original gates for each experiment are shown in Figures 4.21-4.24. These figures illustrate the average ancestral entropy for RandomAlgorithm is either greater or the same as RandomTwoGates and RandomLevelTwoGates. In every experiment RandomAlgorithm replaces all original gates in the circuit.

Boxplots from the 2-Sample t test comparing RandomAlgorithm against RandomLevelTwoGates and RandomTwoGates are displayed in Figures 4.25-4.27. Table 4.7 contains the p values from the 2-Sample t test. RandomAlgorithm does not perform differently than RandomTwoGates and RandomLevelTwoGates for any four gate replacement series experiment.

4.5 Four Gate Replacement C-17 Parallel Circuits

Tables 4.8-4.10 illustrate the relationship between the three and four gate replacement parallel circuits. From this, the RandomAlgorithm four gate replacement experiments generally produce higher entropy values than its three gate counter part.

Table 4.4: RandomLevelTwoGates 3 and 4 Gate Replacement Comparison, Series Circuits

| | Inputs | Average Ancestral Entropy | | | Gates with 3 Ancestors |
|---|-----------|---------------------------|---------------|-------|------------------------|
| | | OutputLevel | Circuit Level | Node | |
| 3 | 5 | 1.18 | .393 | .36 | 63 |
| | 5 ‘Split’ | 1 | .64 | .633 | 57 |
| | 11 | 1.584 | .66 | .71 | 150 |
| 4 | 5 | 1.547 | .6844 | .7796 | 228 |
| | 5 ‘Split’ | 1.4785 | 1.135 | 1.168 | 400 |
| | 11 | .97 | .9132 | .923 | 131 |

Table 4.5: RandomTwoGates 3 and 4 Gate Replacement Comparison, Series Circuits

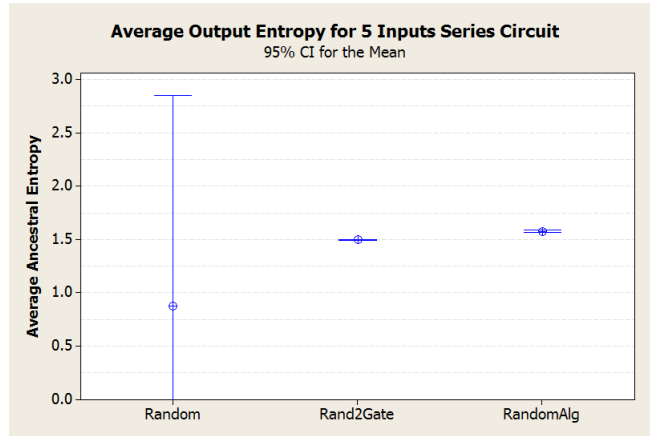
| | Inputs | Average Ancestral Entropy | | | Gates with 3 Ancestors |
|---|-----------|---------------------------|---------------|--------|------------------------|
| | | OutputLevel | Circuit Level | Node | |
| 3 | 5 | 1.516 | .169 | .1567 | 63 |
| | 5 ‘Split’ | 1.56 | 1.126 | 1.118 | 598 |
| | 11 | 1.516 | .858 | .858 | 304 |
| 4 | 5 | 1.501 | .533 | .495 | 11 |
| | 5 ‘Split’ | 1.541 | 1.122 | 1.083 | 338 |
| | 11 | 1.584 | 1.17 | .1.155 | 365 |

Table 4.6: RandomAlgorithm 3 and 4 Gate Replacement Comparison, Series Circuits

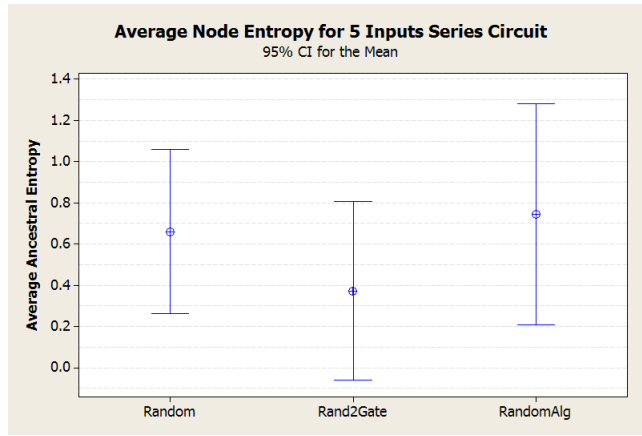
| | Inputs | Average Ancestral Entropy | | | Gates with 3 Ancestors |
|---|-----------|---------------------------|---------------|-------|------------------------|
| | | OutputLevel | Circuit Level | Node | |
| 3 | 5 | 1.585 | 1.356 | 1.024 | 522 |
| | 5 ‘Split’ | 1.581 | 1.585 | 1.107 | 467 |
| | 11 | 1.584 | 1.265 | 1.227 | 687 |
| 4 | 5 | 1.578 | .849 | .869 | 281 |
| | 5 ‘Split’ | 1 | 1.056 | 1.057 | 209 |
| | 11 | 1 | .939 | .994 | 176 |

Table 4.7: P-Values from 2-Sample t Test comparing RandomAlgorithm against RandomTwoGates and RandomLevelTwoGates Using Series Circuits with a 4 Gate Replacement

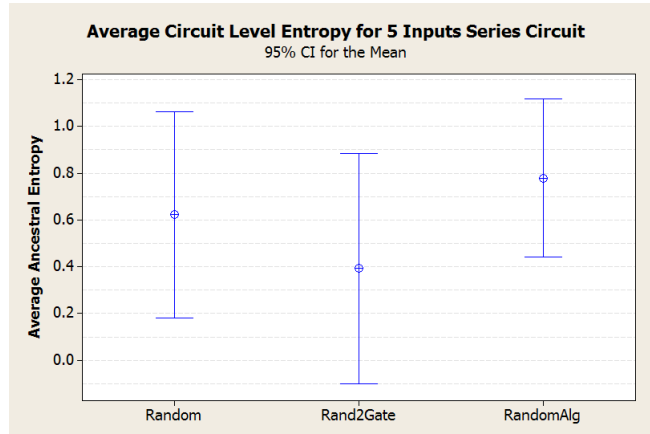
| Entropy | 5 Input | 5 ‘Split’ Input | 11 Input |
|---------------|---------|-----------------|----------|
| Output Level | .316 | .793 | .715 |
| Node | .181 | .343 | .844 |
| Circuit Level | .085 | .548 | .485 |



(a) Output Ancestral Entropy Results

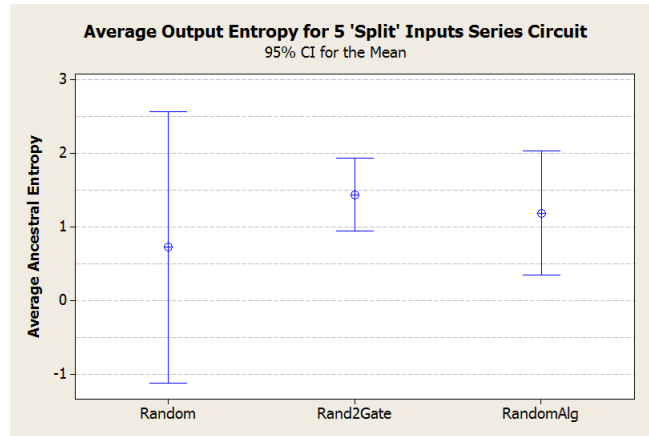


(b) Node Ancestral Entropy Results

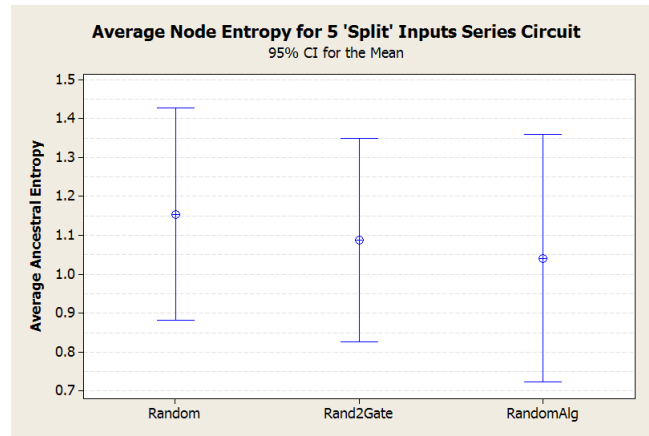


(c) Circuit Level Ancestral Entropy Results

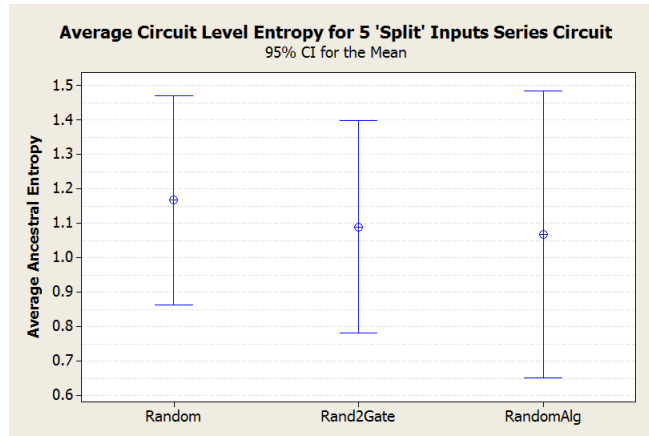
Figure 4.21: Average Ancestral Entropy Intervals for 5 Input Series Circuit with a 4 Gate Replacement (a), (b) and (c)



(a) Output Ancestral Entropy Results

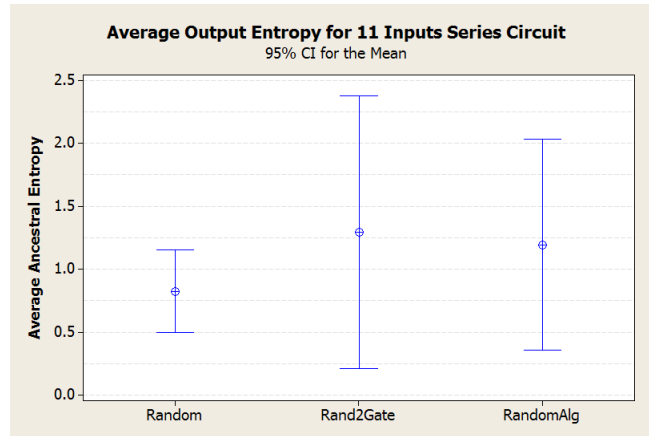


(b) Node Ancestral Entropy Results

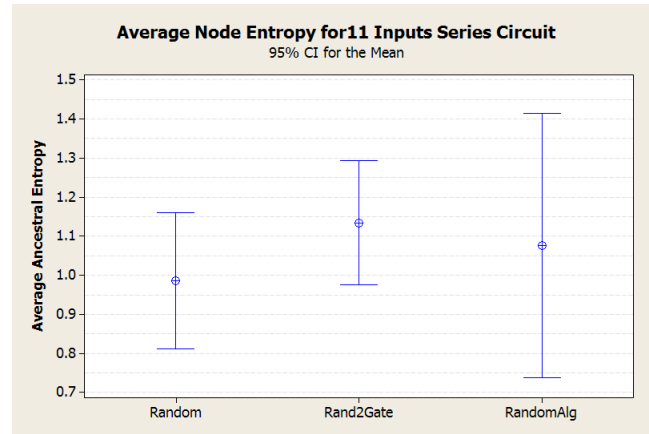


(c) Circuit Level Ancestral Entropy Results

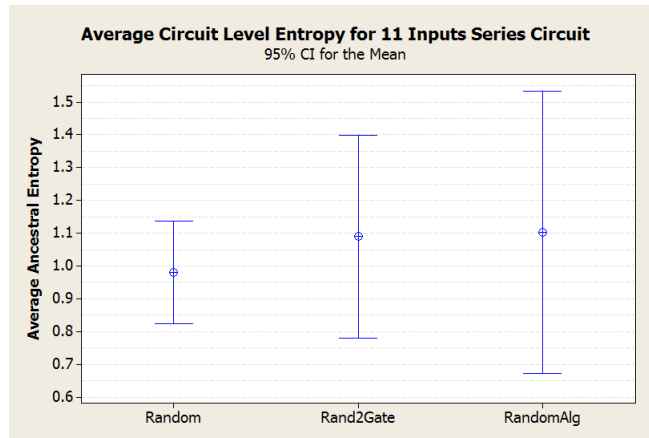
Figure 4.22: Average Ancestral Entropy Intervals for 5 'Split' Input Series Circuit with a 4 Gate Replacement (a), (b) and (c)



(a) Output Ancestral Entropy Results

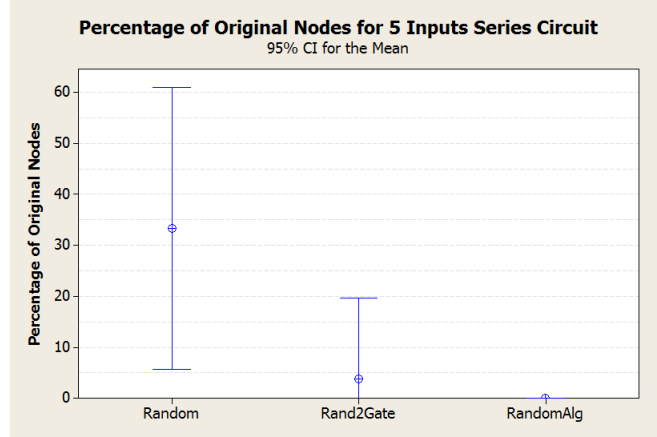


(b) Node Ancestral Entropy Results

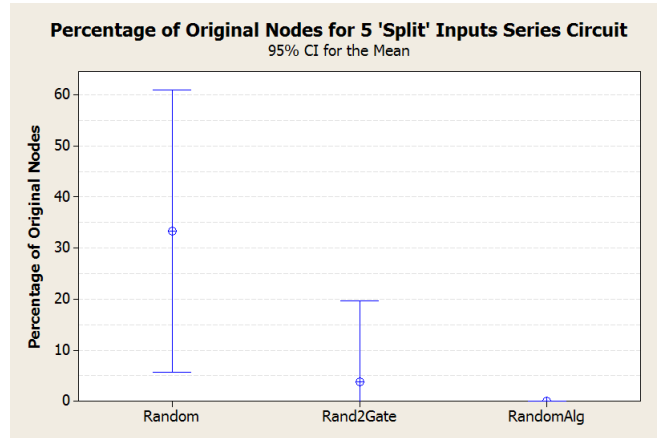


(c) Circuit Level Ancestral Entropy Results

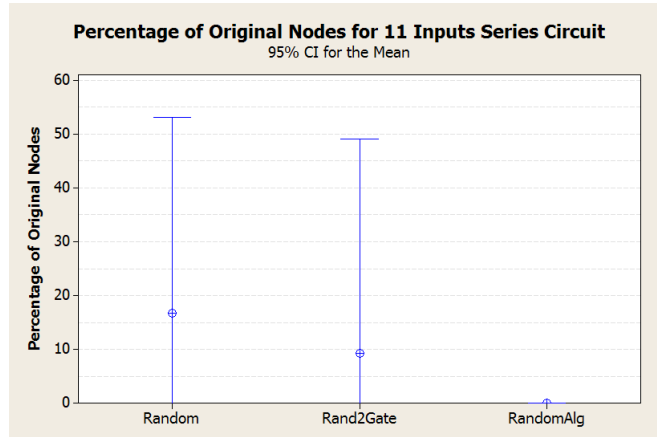
Figure 4.23: Average Ancestral Entropy Intervals for 11 Input Series Circuit with a 4 Gate Replacement (a), (b) and (c)



(a) Percentage of Original gates in 5 Input Circuits with a 4 Gate Replacement

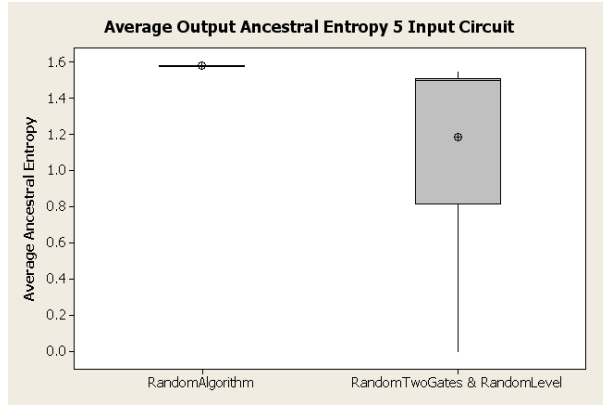


(b) Percentage of Original gates in 5 'Split' Input Circuits

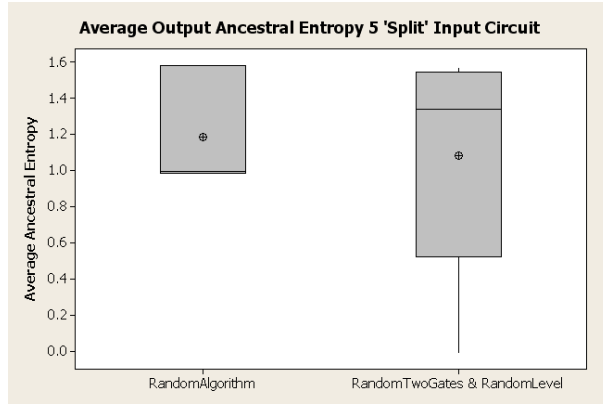


(c) Percentage of Original gates in 11 Input Circuits

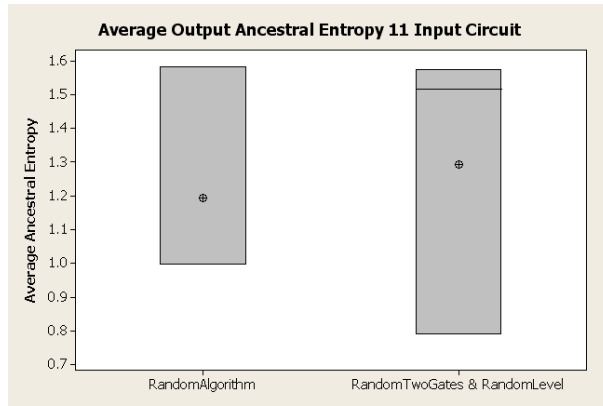
Figure 4.24: Percentage of Original Gates in Series Circuits with a 4 Gate Replacement (a), (b) and (c)



(a) Boxplot for 5 Input Series Circuits

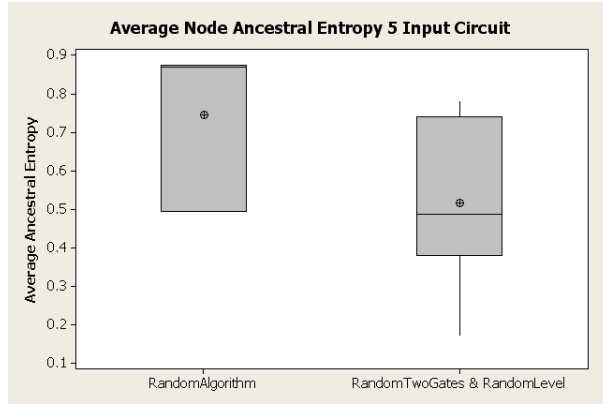


(b) Boxplot for 5 Split Input Series Circuits

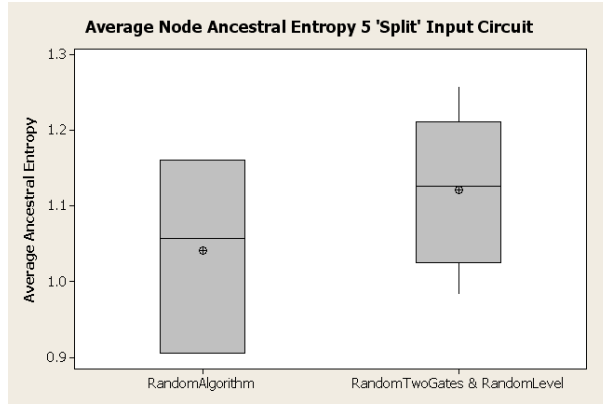


(c) Boxplot for 11 Input Series Circuits

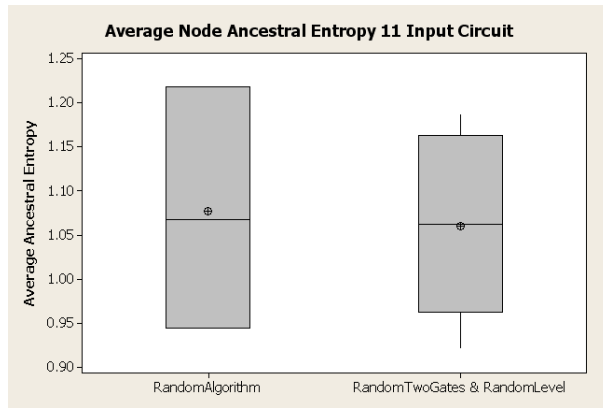
Figure 4.25: Boxplot from 2-Sample t Test of Average Output Ancestral Entropy with Random Algorithms on Series Circuits with a 4 Gate Replacement (a), (b) and (c)



(a) Boxplot for 5 Input Series Circuits

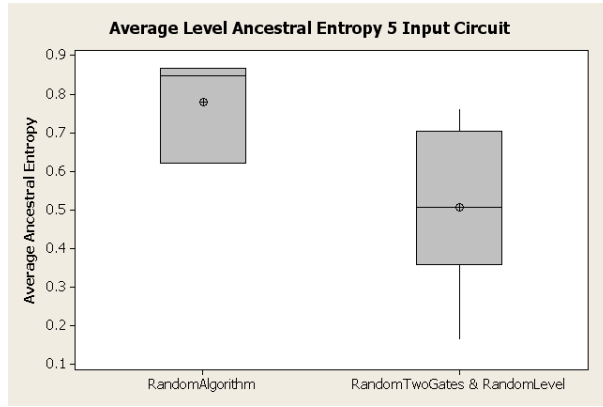


(b) Boxplot for 5 Split Input Series Circuits

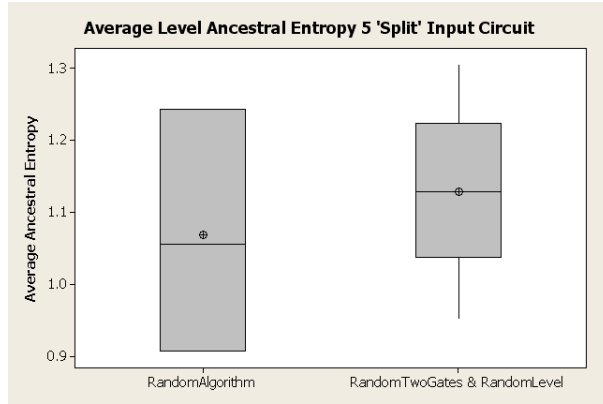


(c) Boxplot for 11 Input Series Circuits

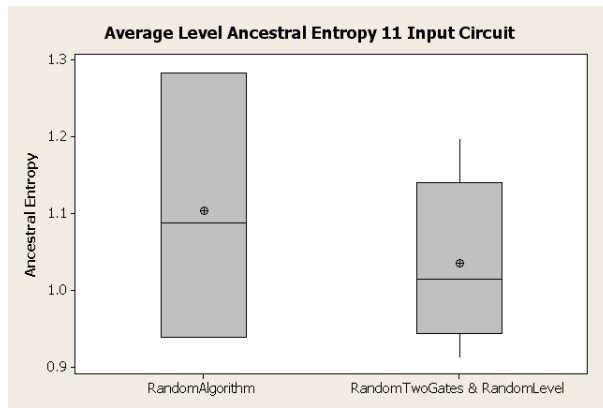
Figure 4.26: Boxplot from 2-Sample t Test of Average Node Ancestral Entropy with Random Algorithms on Series Circuits with a 4 Gate Replacement (a), (b) and (c)



(a) Boxplot for 5 Input Series Circuits



(b) Boxplot for 5 Split Input Series Circuits



(c) Boxplot for 11 Input Series Circuits

Figure 4.27: Boxplot from 2-Sample t Test of AverageLevel Ancestral Entropy with Random Algorithms on Series Circuits with a 4 Gate Replacement (a), (b) and (c)

Comparing the four gate replacement charts in Appendix C with the three gate replacement charts in Appendix B it appears the random algorithm four gate replacement produces higher ancestral entropy than the three gate replacement experiments.

Table 4.11 contains the p values from the 2-Sample t test comparing RandomAlgorithm against RandomTwoGates and RandomLevelTwoGates. RandomAlgorithm is significantly different than the other two algorithms in 4 of the 6 values.

Table 4.8: RandomLevelTwoGates 3 and 4 Gate Replacement Comparison, Parallel Circuits

| | Inputs | Average Ancestral Entropy | | | Gates with Max # Ancestors |
|---|------------|---------------------------|---------------|-------|-------------------------------|
| | | OutputLevel | Circuit Level | Node | |
| 3 | Shared | 1.9635 | .9893 | .9647 | 153 |
| | Individual | .9032 | .7322 | .7788 | 253 |
| 4 | Shared | .9685 | .996 | 1.049 | 87 |
| | Individual | .5715 | .8584 | .8633 | 54 |

Table 4.9: RandomTwoGates 3 and 4 Gate Replacement Comparison, Parallel Circuits

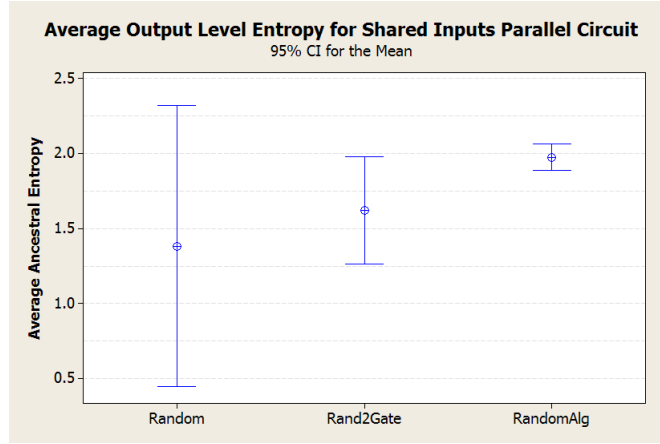
| | Inputs | Average Ancestral Entropy | | | Gates with Max # Ancestors |
|---|------------|---------------------------|---------------|--------|-------------------------------|
| | | OutputLevel | Circuit Level | Node | |
| 3 | Shared | 1.344 | .8222 | .8128 | 313 |
| | Individual | 1.774 | .7736 | .725 | 11 |
| 4 | Shared | 1.684 | .5257 | .5536 | 3 |
| | Individual | 1.0383 | 1.233 | 1.2185 | 418 |

Table 4.10: RandomAlgorithm 3 and 4 Gate Replacement Comparison, Parallel Circuits

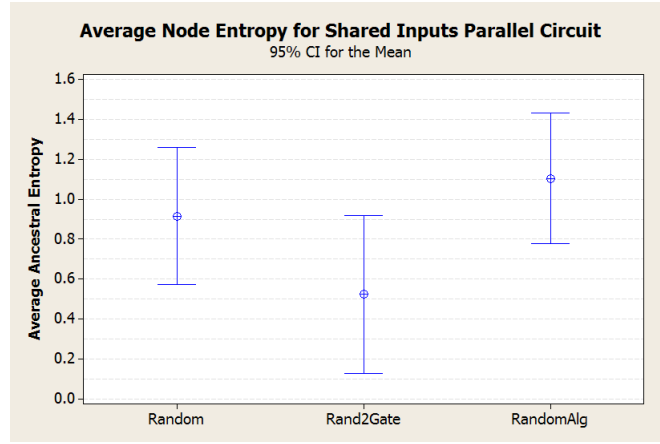
| | Inputs | Average Ancestral Entropy | | | Gates with Max # Ancestors |
|---|------------|---------------------------|---------------|--------|-------------------------------|
| | | OutputLevel | Circuit Level | Node | |
| 3 | Shared | 1.9932 | 1.206 | 1.0536 | 271 |
| | Individual | 1.5814 | .9046 | .9974 | 528 |
| 4 | Shared | 1.999 | 1.0353 | .9676 | 203 |
| | Individual | 1.568 | 1.0659 | 1.1215 | 373 |

Table 4.11: P-Values from 2-Sample t Test comparing RandomAlgorithm against RandomTwoGates and RandomLevelTwoGates Using Parallel Circuits with a 4 Gate Replacement

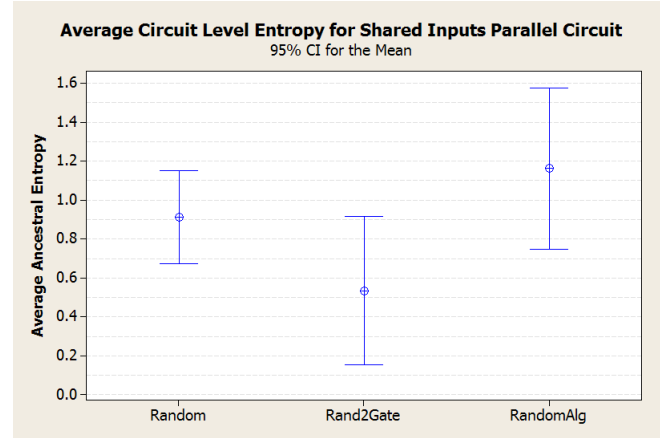
| Entropy | Shared Input | Individual Input |
|---------------|--------------|------------------|
| Output Level | .028 | .003 |
| Node | .001 | .704 |
| Circuit Level | .001 | .443 |



(a) Output Ancestral Entropy Results

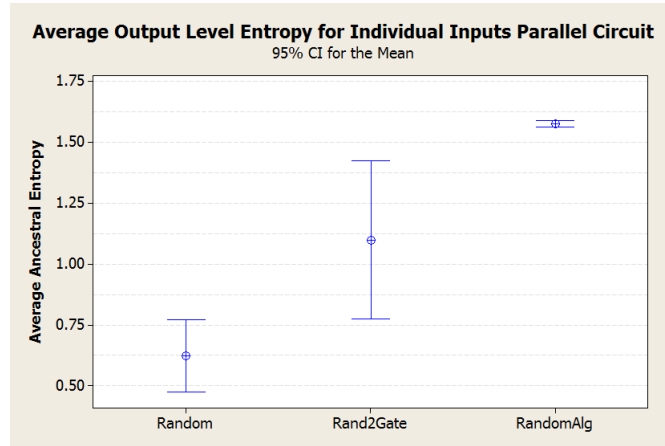


(b) Node Ancestral Entropy Results

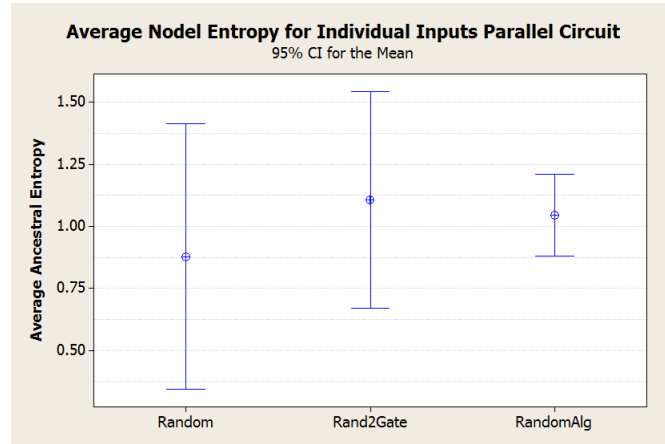


(c) Circuit Level Ancestral Entropy Results

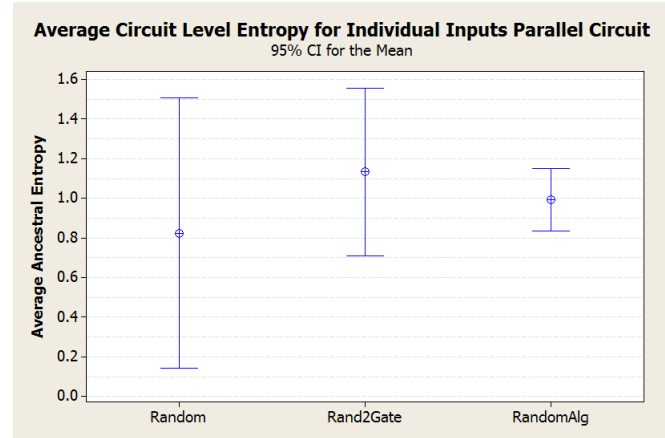
Figure 4.28: Average Ancestral Entropy Intervals for Shared Input Parallel Circuit with a 4 Gate Replacement (a), (b) and (c)



(a) Output Ancestral Entropy Results

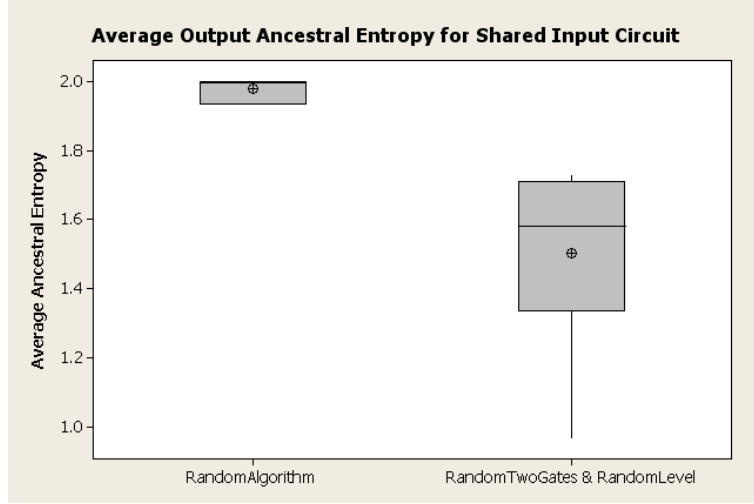


(b) Node Ancestral Entropy Results

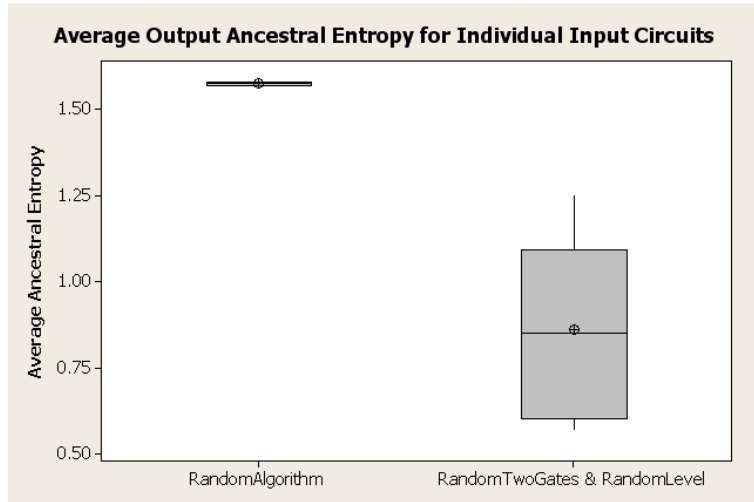


(c) Circuit Level Ancestral Entropy Results

Figure 4.29: Average Ancestral Entropy Intervals for Individual Input Parallel Circuit with a 4 Gate Replacement (a), (b) and (c)

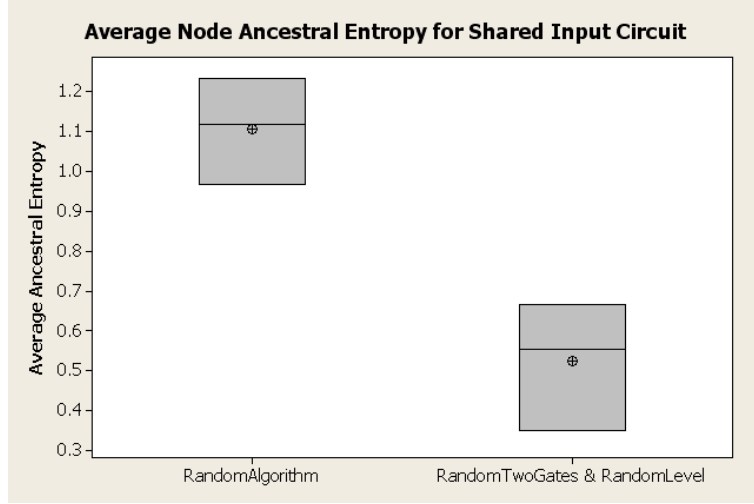


(a) Boxplot for Shared Input Parallel Circuits

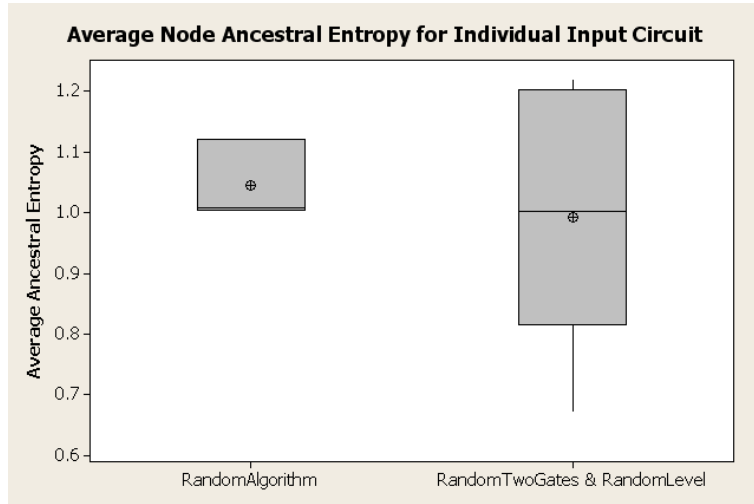


(b) Boxplot for Individual Input Parallel Circuits

Figure 4.30: Boxplot from 2-Sample t Test of Average Output Ancestral Entropy with Random Algorithms on Four Gate Replacement Parallel Circuits (a) and (b)

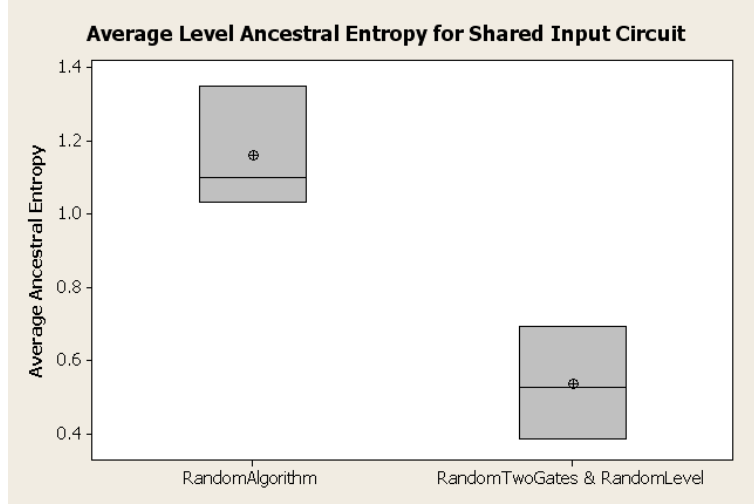


(a) Boxplot for Shared Input Parallel Circuits

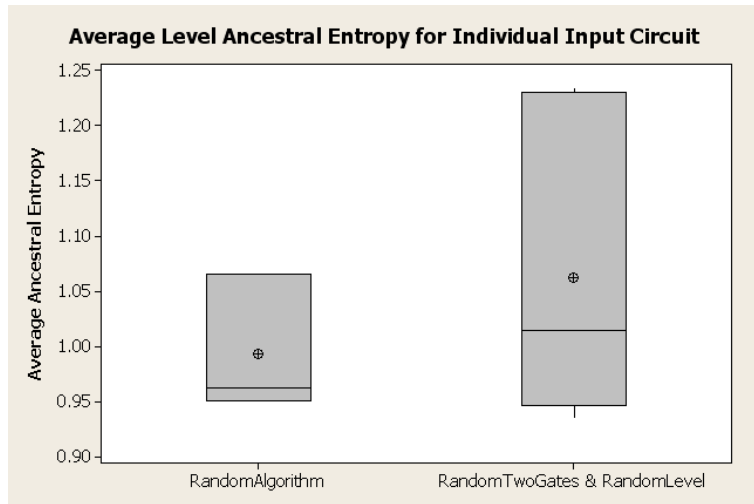


(b) Boxplot for Individual Input Parallel Circuits

Figure 4.31: Boxplot from 2-Sample t Test of Average Node Ancestral Entropy with Random Algorithms on Four Gate Replacement Parallel Circuits (a) and (b)



(a) Boxplot for Shared Input Parallel Circuits



(b) Boxplot for Individual Input Parallel Circuits

Figure 4.32: Boxplot from 2-Sample t Test of Average Level Ancestral Entropy with Random Algorithms on Four Gate Replacement Parallel Circuits (a) and (b)

Table 4.12: Reduction Results

| | | | | |
|--------------------------------------|--------|------|--|-----------------------|
| Iteration: | 300 | 100 | 300 | 100 |
| RandomAlgorithm | $H(P)$ | | Gate Reduction(<i>original</i> \rightarrow <i>reduced</i>) | |
| 2-4 Duplicate and Redundant Gates | | | | |
| Series | .641 | .646 | 617 \rightarrow 175 | 218 \rightarrow 61 |
| Parallel | .78 | .32 | 611 \rightarrow 101 | 218 \rightarrow 32 |
| 2-4, No Duplicate or Redundant Gates | | | | |
| Series | .87 | .67 | 614 \rightarrow 473 | 218 \rightarrow 168 |
| Parallel | 1.12 | .71 | 618 \rightarrow 414 | 218 \rightarrow 145 |

4.6 Validation

An immediate independent validation tool is Kim’s [9] reduction software. This reduction tool was designed knowing how CORGI generates gate replacements. The expectation is circuits with higher ancestral entropy will reduce less than circuits with low ancestral entropy. Also, given a circuit with independent components (i.e., the independent input parallel circuit) and a circuit variant that reduces to independent components the ancestral entropy should be less than the ancestral entropy of a circuit that does not reduce to independent components.

Table 4.12 displays the results of reduction on the two series and parallel circuit variants created using RandomAlgorithm with a two gate selection and a four gate replacement. Circuits with higher entropy do reduce less. Both the 100 and 300 iteration series circuit allowing duplicate inputs and redundant gates have a 72% reduction and a 23% reduction when they are not allowed. Both the 100 and 300 iteration parallel circuit allowing duplicate inputs and redundant gates have a reduction of approximately 85%. When duplicate inputs and redundant gates are not allowed there is only an approximate reduction of 33%. This supports the expectation that circuits with higher ancestral entropy reduce less.

Circuits also should require less iterations to be performed before the reducer fails to separate the circuit into independent components. Starting with the circuit in Figure 4.33 several circuit variants are created allowing duplicate inputs and redundant gates. As displayed in Figures 4.34 and 4.35, at the 19th iteration of creating

the variant ($H(P) = .0247$) the circuit is reduced to three independent components and at the 28th iteration ($H(P) = .078$) the reducer cannot separate the variant into multiple independent components. When disallowing duplicate inputs and redundant gates the reducer separates the 18th iteration ($H(P) = .178$) into three independent components and cannot separate it after the 24th iteration ($H(P) = .212$) and are illustrated in Figures 4.36 and 4.37.

Although this is just one example it appears that circuits with higher ancestral entropy are more difficult to reduce to individual components than circuits with lower entropy. More experiments must be performed to verify this.

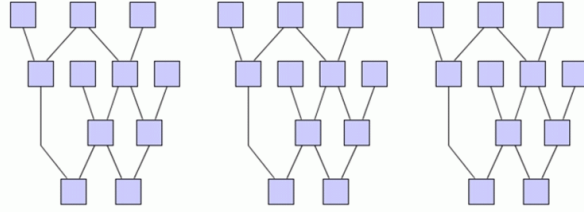


Figure 4.33: Baseline Individual Input Parallel C-17 Circuit

4.7 Summary

These results show that RandomAlgorithm creates circuit variants with higher average ancestral entropy values at the outputs, circuit levels, and for every node in the circuit than the other CORGI algorithms. RandomTwoGates performed nearly as well or better than RandomAlgorithm in some cases. More experiments need to be performed to determine if one is truly better than the other based on whether removing original sub-circuits in early iterations will cause the ancestral entropy values to be lower. However, these two algorithms outperform the other CORGI algorithms in terms of providing component hiding characteristics to a circuit variant.

High Ancestral entropy and the ability to reduce a circuit appear to have a direct correlation. Circuits with higher ancestral entropy were not reduced as well as other and were more difficult to separate the parallel circuit into independent components.

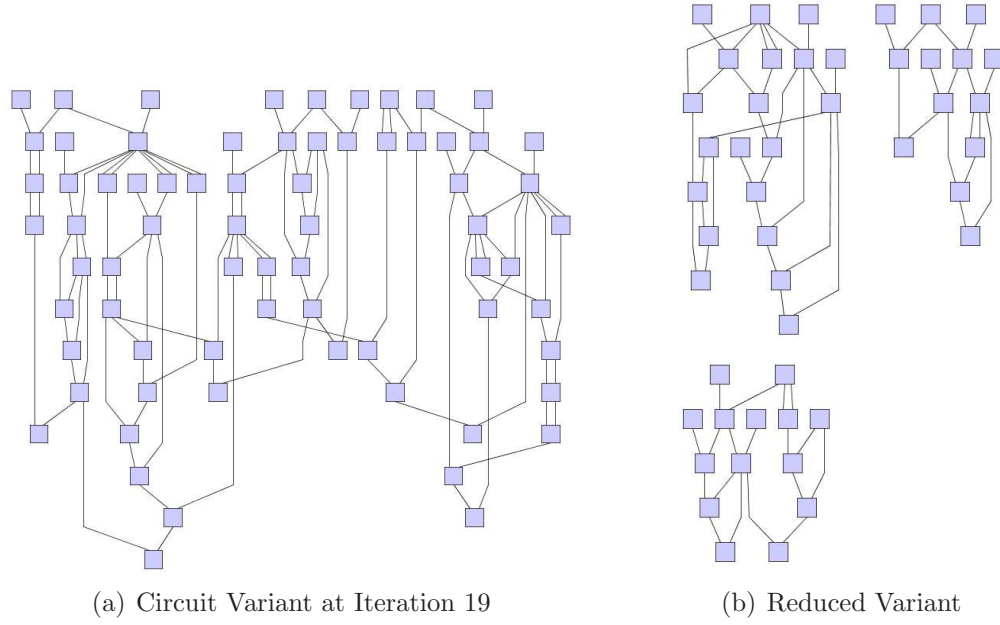


Figure 4.34: Reducing C-17 Parallel Circuit with Duplicate Inputs and Redundant Gates to Independent Components
(a) and (b)

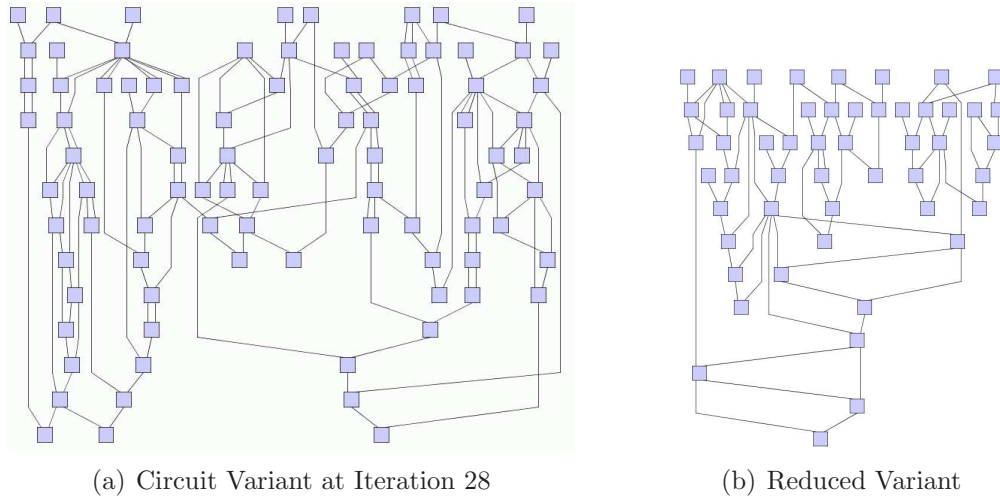
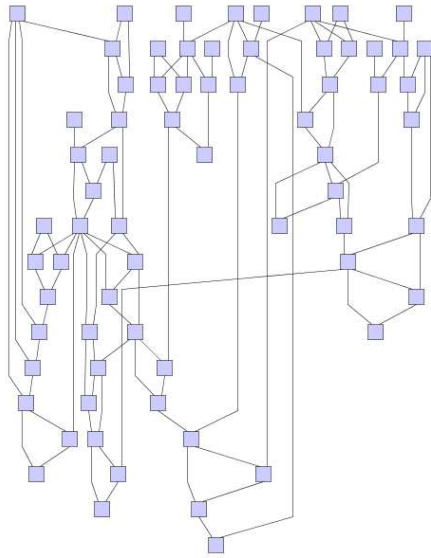
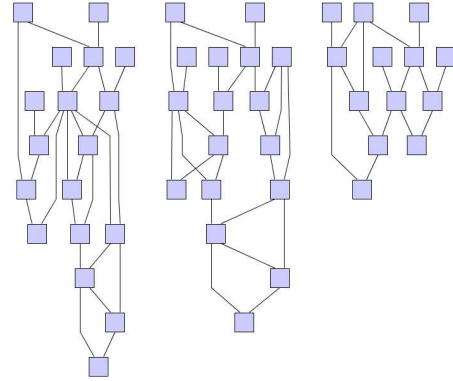


Figure 4.35: Reducing C-17 Parallel Circuit with Duplicate Inputs and Redundant Gates One Iteration Before Component Separation
(a) and (b)

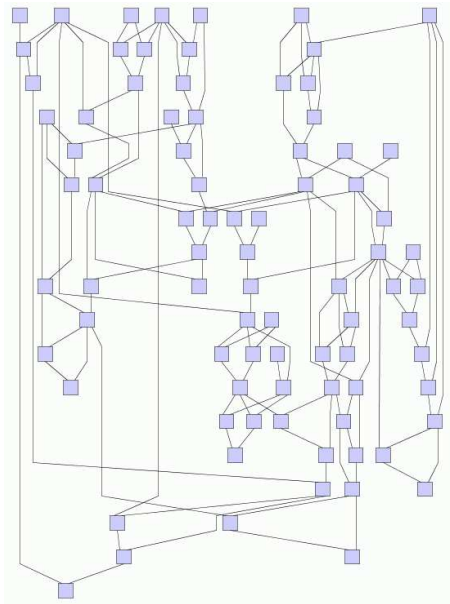


(a) Circuit Variant at Iteration 18

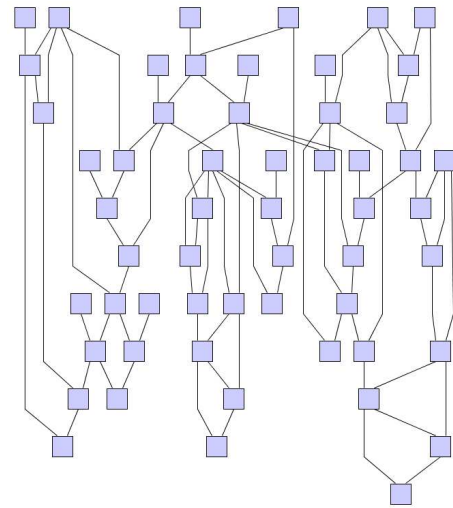


(b) Reduced Variant

Figure 4.36: Reducing C-17 Parallel Circuit with No Duplicate Inputs or Redundant Gates to Independent Components
(a) and (b)



(a) Circuit Variant at Iteration 25



(b) Reduced Variant

Figure 4.37: Reducing C-17 Parallel Circuit with No Duplicate Inputs or Redundant Gates One Iteration Before Component Separation
(a) and (b)

V. Conclusions

This chapter discusses the contribution of this research and future work.

5.1 *Goals and Hypothesis*

The terms component hiding and ancestral entropy are defined and described using ancestral entropy as a measure of component hiding. The iterative selection and replacement algorithms are assumed to provide measurable component hiding properties, resulting in the circuit transformations not being easily undone by a circuit reducer (or a reverse engineer). Series and parallel combinational circuits with different configurations are compared and the component hiding properties are identified.

5.2 *Contributions*

5.2.1 Component Hiding Effectiveness. Ancestral entropy measures the uncertainty a node or gate in a circuit derived from a specific sub-circuit. The ancestral entropy is calculated for each node, the average at each level in the circuit, and the average at the output nodes. These entropy values provide useful information to compare the effectiveness of the selection and replacement algorithms in terms of component hiding. It also provides the data necessary for creating the colors in the graphs used in this research.

5.2.2 Introduction of the Colored Graphs. The colored graphs are an improvement over the current graphs output by the CORGI. The use of colors, shape, numbers and ancestry values allow for a quick, visual description of what each iteration of an algorithm is doing. Circuit merging and removal is easily identified by the introduction or removal of a color from the graph. The graph coloring process also creates a file identifying the iteration number when a new color is introduced in the graph *and* when a color is removed from the graph. It defines what color nodes are present, how many nodes of each color, and the total number of nodes for each iteration. Providing a shape for each node provides an immediate indication of the

gate operation and aids in the identification of buffer circuits that could easily be reduced. Adding the node number, gate type, and ancestry information to each node label adds a trace capability back to the BENCH and ancestry files to verify proper operation of the selection and replacement algorithm, the ancestry algorithm, and the entropy algorithm developed in this research.

5.3 Future Work

5.3.1 Color Graph Improvements. The appearance of the colored graphs could be further improved with the implementation of a red, blue and green (RBG) or similar coloring scheme. Using the ancestry values as weighted color values will provide graphs that show color variations and visually indicate the ancestors, and the weight of the ancestry, of each node in the circuit. The current implementation does not allow coloring of the optimized circuits when using the CORGI circuit reducing program. This improvement will provide a visual representation of the reduced circuits and will be useful in analyzing them.

5.3.2 Execute at Runtime. The current implementation for computing ancestry, entropy and coloring the graphs are all done by reading files created during the initial execution of the algorithm. Although this is done intentionally for use on previous experiments, it is worthwhile to perform these functions during runtime. It is also worthwhile to add the option to include or not include computing these functions during the experiment, due to the increase in runtime.

5.3.3 Modify FixedLevelTwoGates Algorithm. The fixed level in FixedLevelTwoGates algorithm is selectable by the user. It is beneficial, especially since RandomAlgorithm is likely to be the best algorithm for component hiding, to have the ability to set the fixed level to the level below the input level. A modification to have the first gate selected from the $level_{max} - 1$ and the second gate from either $level_{max} - 1$ or $level_{max} - 2$ is needed. This will cause the RandomAlgorithm to select

gates for replacement in a more uniform manner by not having two algorithms make replacements at or near the output level. Allowing replacements to be made near the input level will most likely provide better component hiding properties than the current implementation

5.3.4 Validation. To validate the reduction tool works properly and that ancestral entropy is a viable measurement for component hiding we need to test the circuit variants against commercial circuit reducing tools. Kim’s algorithm [9] may or may not provide enough reduction to determine if the four gate replacement experiments provide component hiding attributes. In an ideal setting reverse engineers would be used to determine the ease of reducing any circuit variants and identifying weaknesses in the creation of the circuit variants.

Appendix A. Three Gate Replacement Series Circuit Variant Graphs and Charts

This appendix contains the graphs and charts for the series circuits discussed in Chapter IV. These charts and graphs are from one experiment only and are provided to show trends of each algorithm.

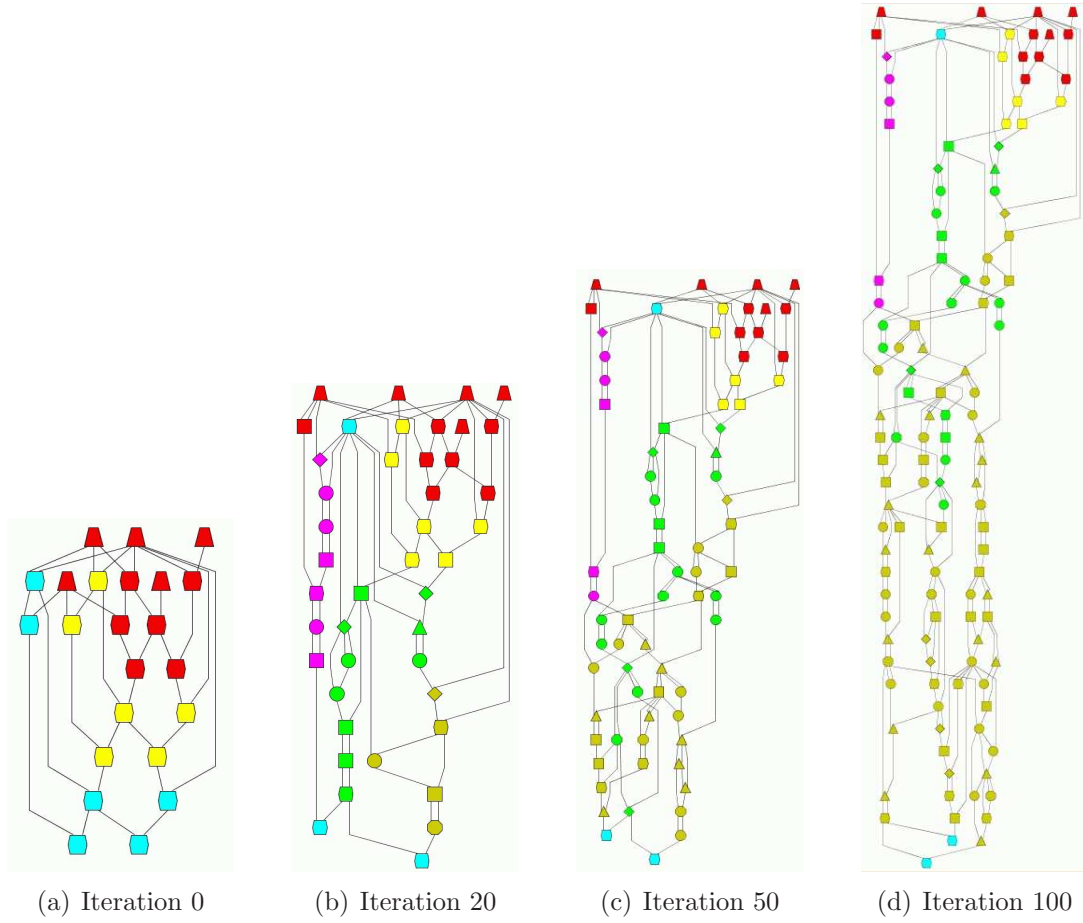
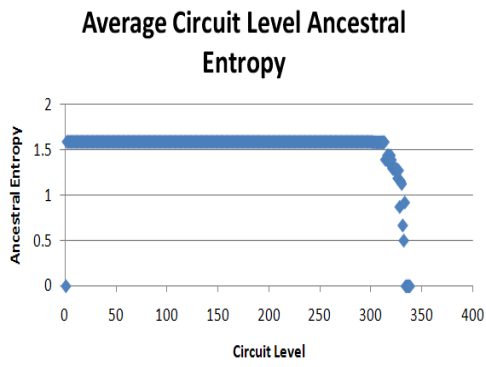
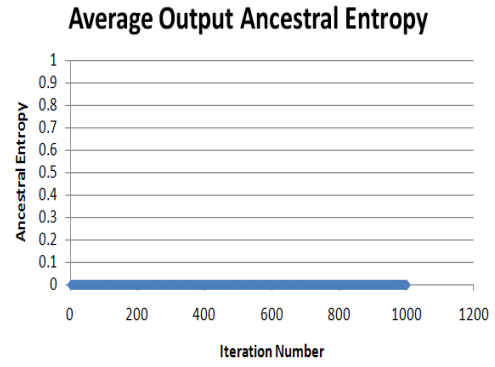


Figure A.1: FixedLevelTwoGates 5 Input C-17 Series Variant Circuits
(a),(b),(c) and (d)

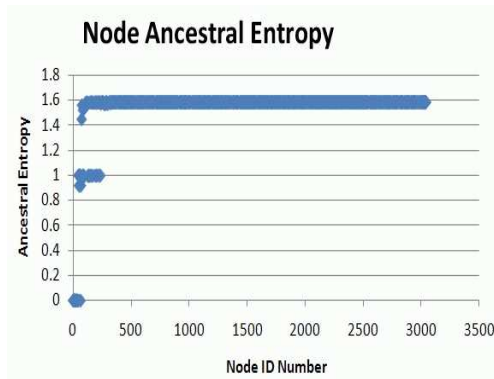
input



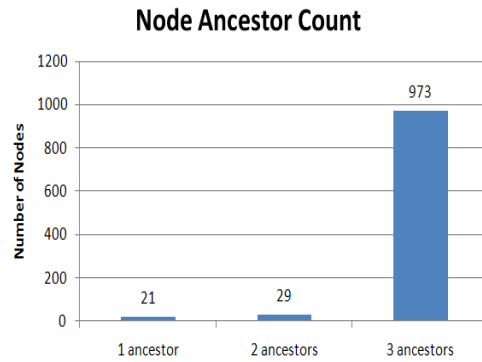
(a)



(b)



(c)



(d)

Figure A.2: FixedLevelTwoGates 5 Input C-17 Series - Iteration 1000
(a),(b),(c) and (d)

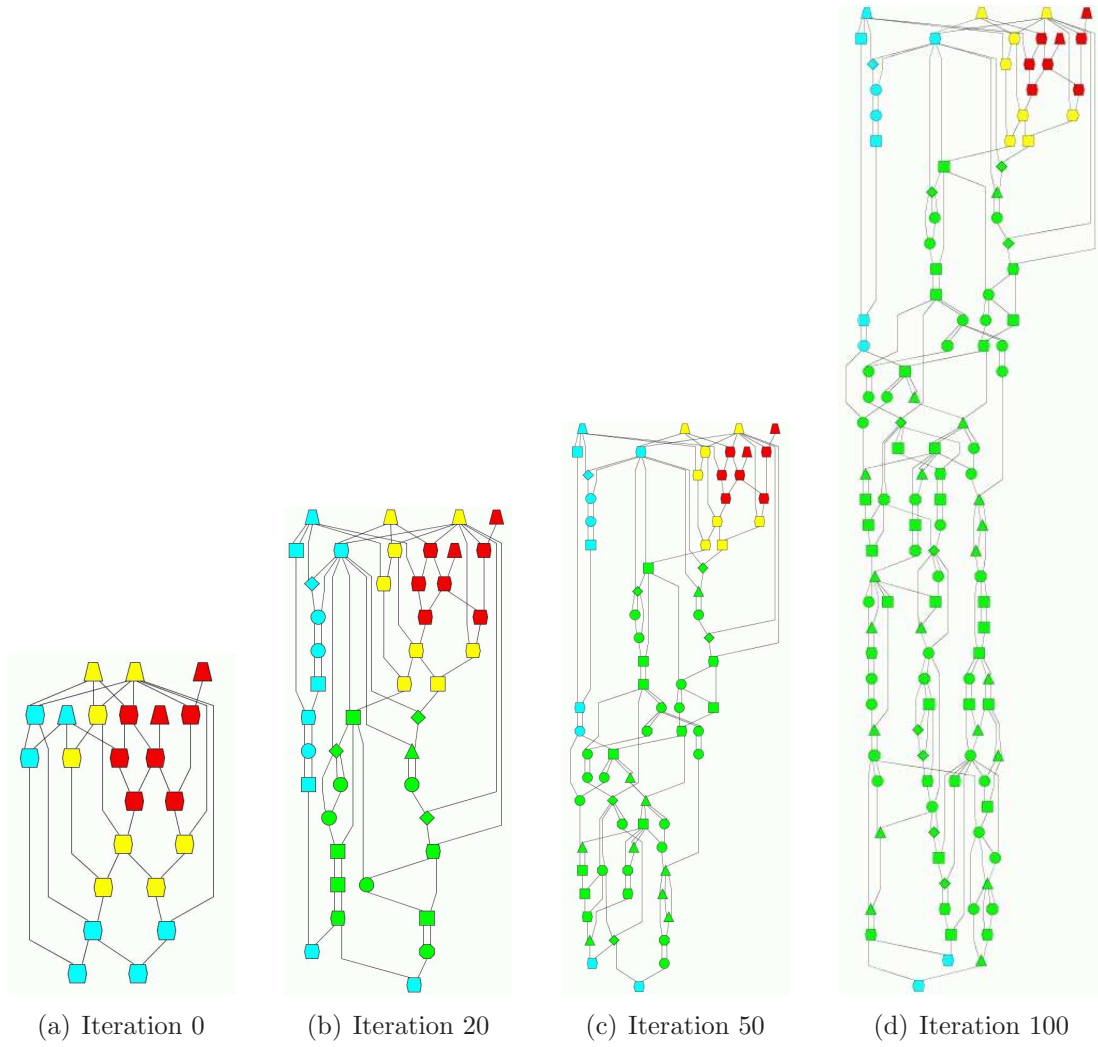


Figure A.3: FixedLevelTwoGates 5 'Split' Input C-17 Series Variant Circuits (a),(b),(c) and (d)

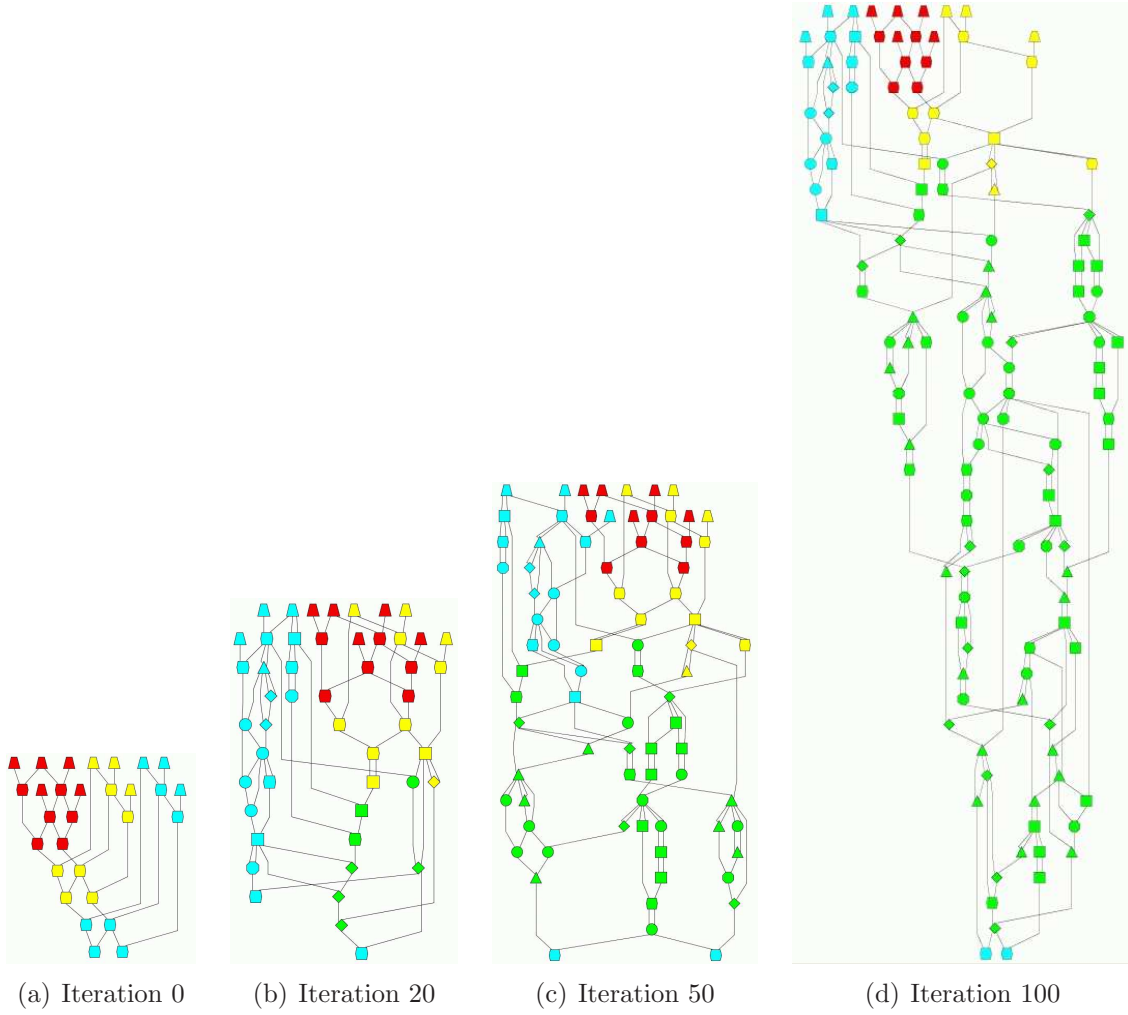
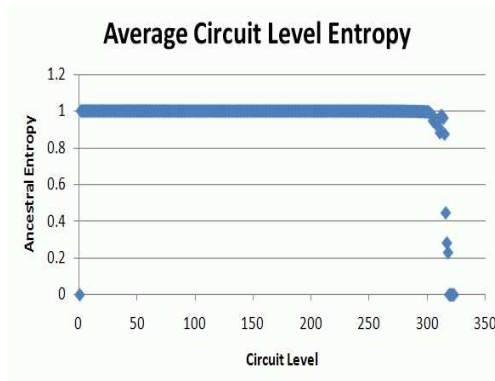
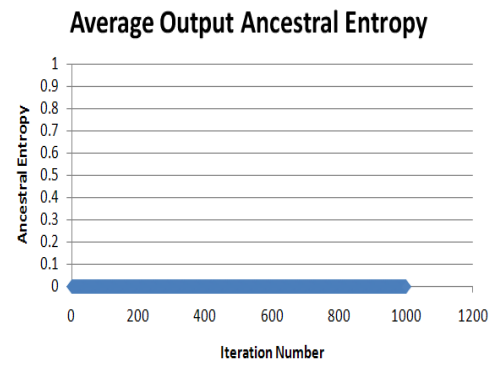


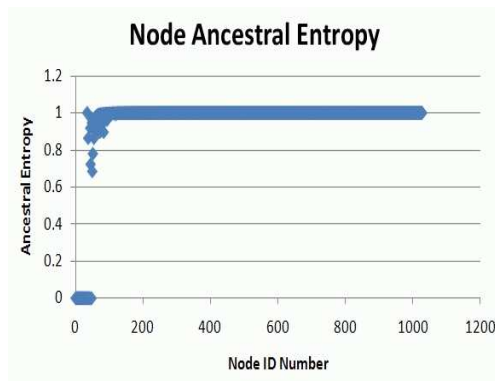
Figure A.5: FixedLevelTwoGates 11 Input C-17 Series Variant Circuits
(a),(b),(c) and (d)



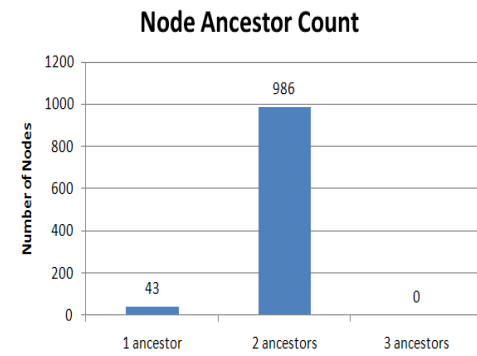
(a)



(b)



(c)



(d)

Figure A.6: FixedLevelTwoGates 11 Input C-17 Series - Iteration 1000
(a),(b),(c) and (d)

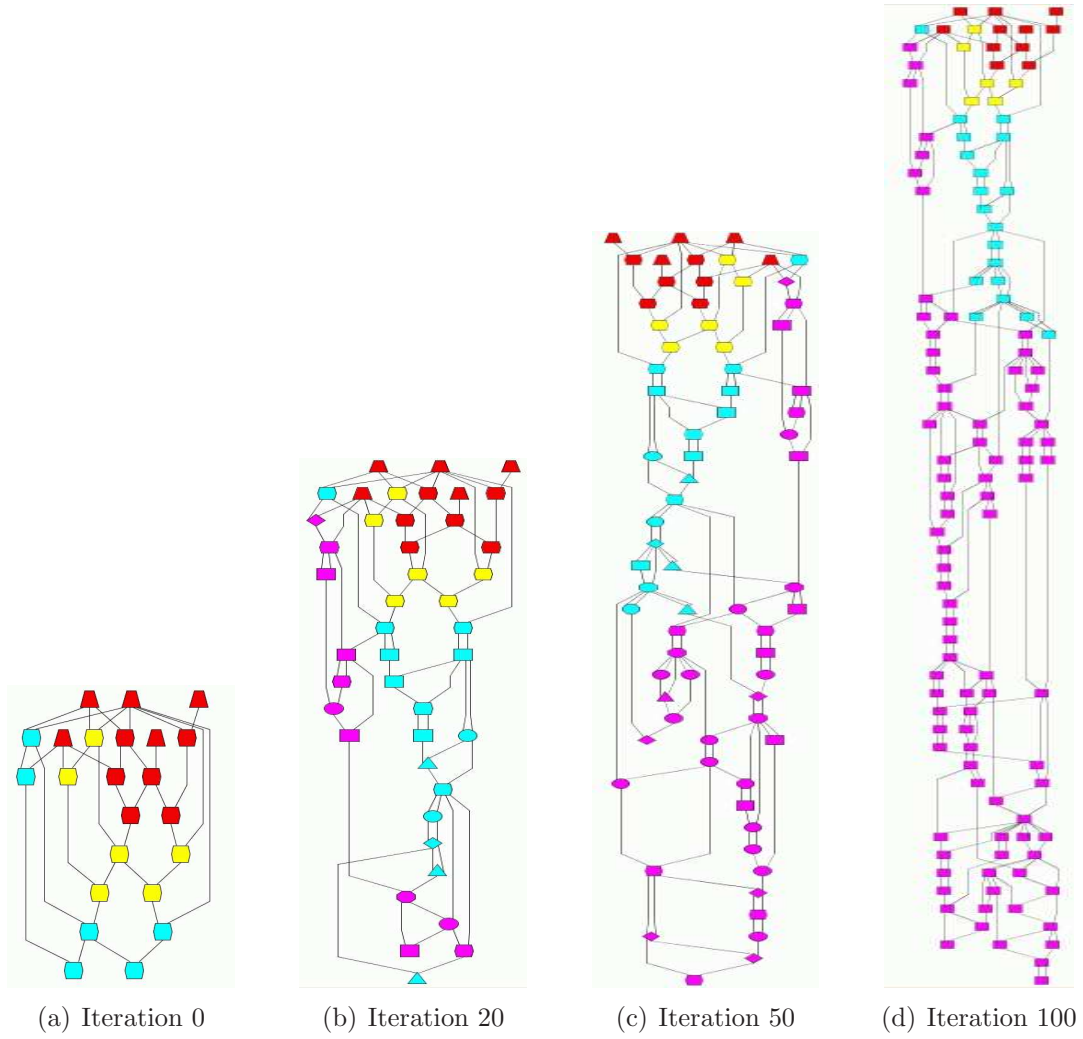
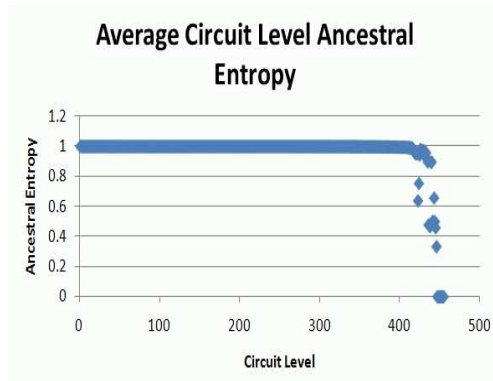
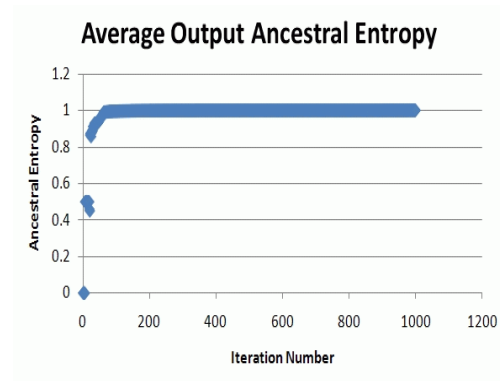


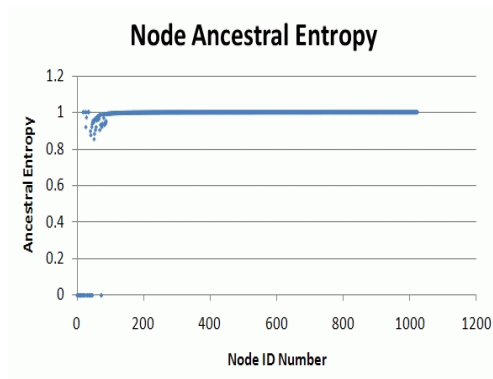
Figure A.7: OutputLevelTwoGates 5 Input C-17 Series Variant Circuits
(a),(b),(c) and (d)



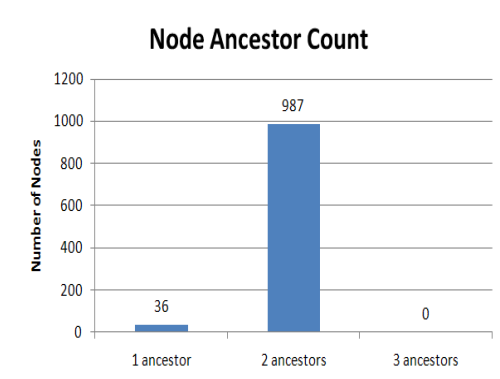
(a)



(b)



(c)



(d)

Figure A.8: OutputLevelTwoGates 5 Input C-17 Series - Iteration 1000
(a),(b),(c) and (d)

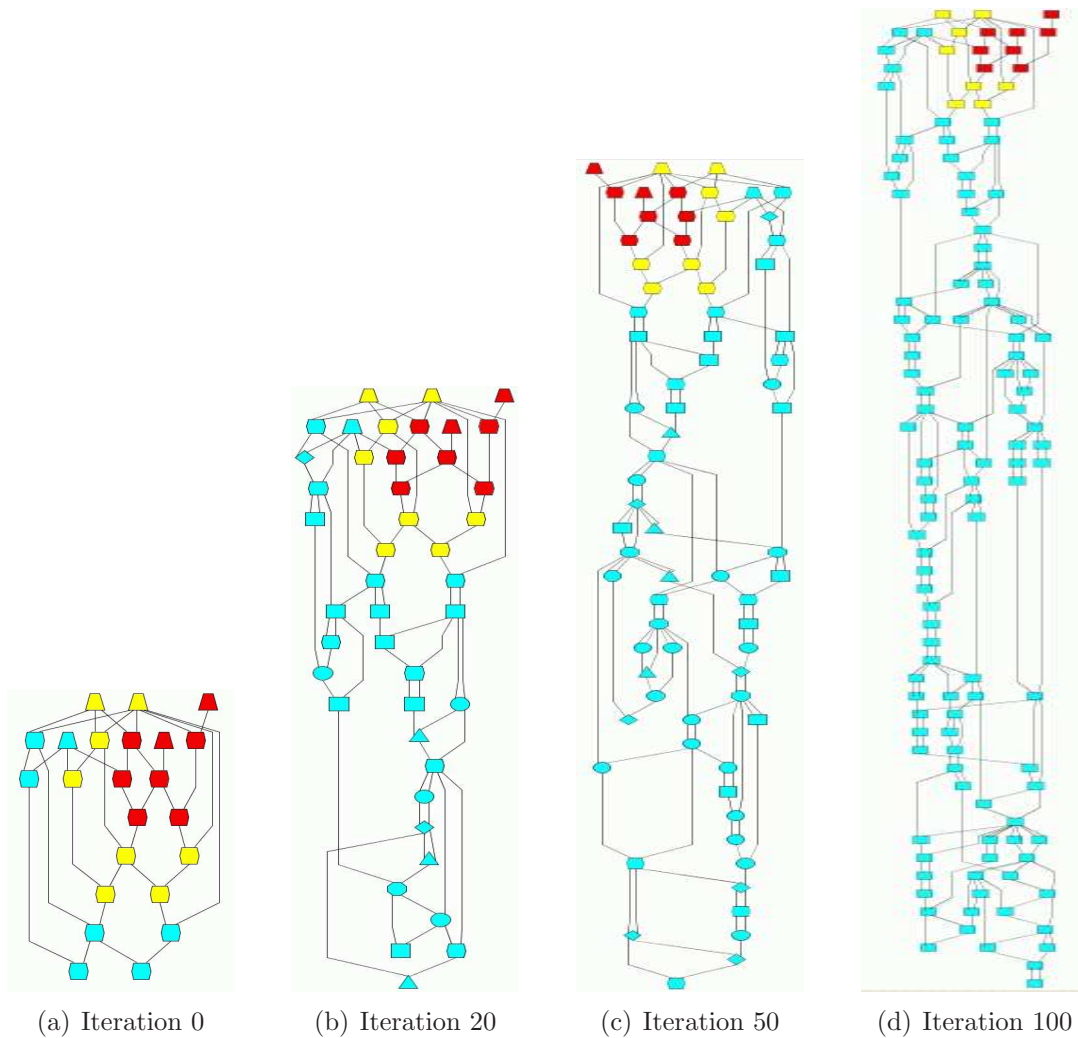
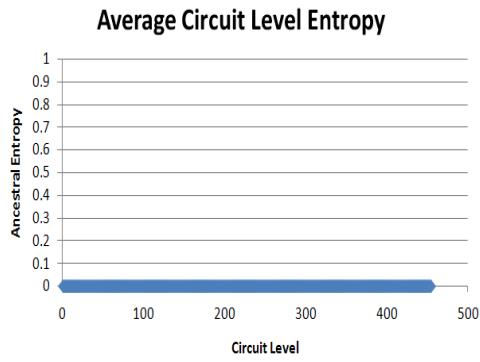
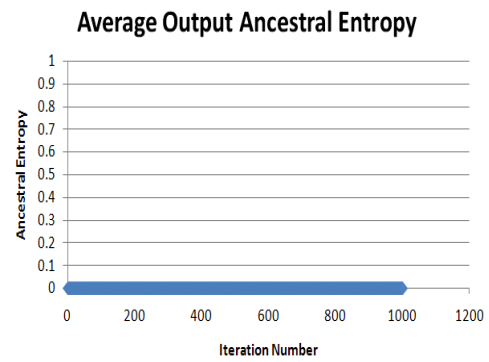


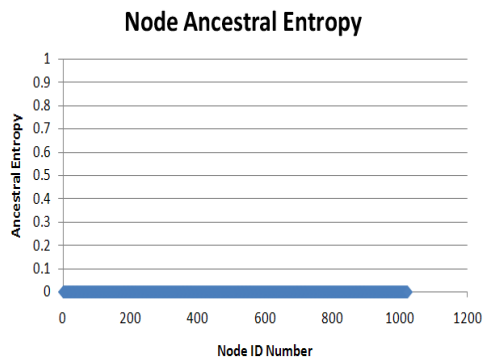
Figure A.9: OutputLevelTwoGates 5 ‘Split’ Input C-17 Series Variant Circuits
(a),(b),(c) and (d)



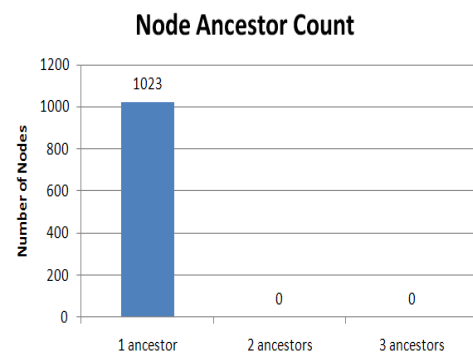
(a)



(b)



(c)



(d)

Figure A.10: OutputLevelTwoGates 5 ‘Split’ Input C-17 Series - Iteration 1000
(a),(b),(c) and (d)

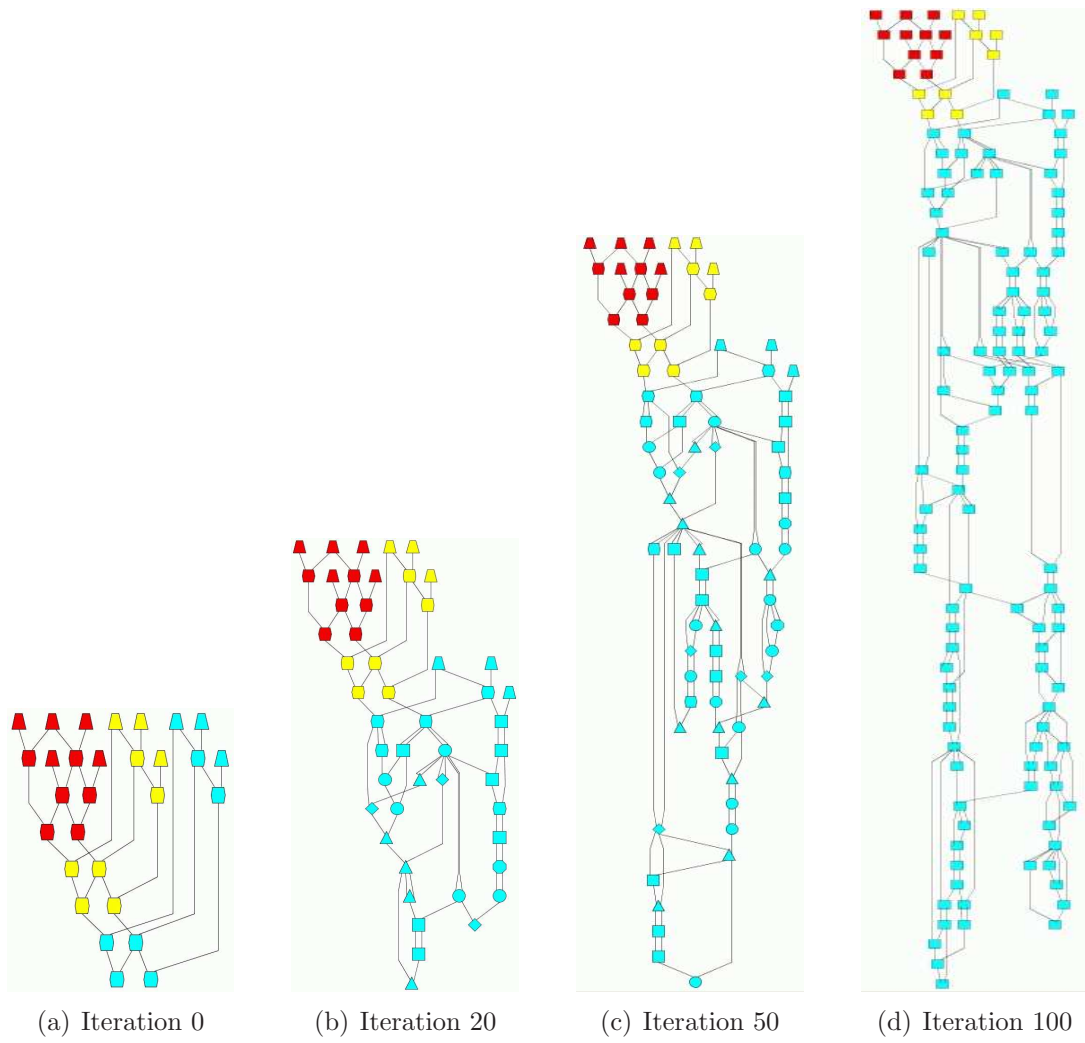
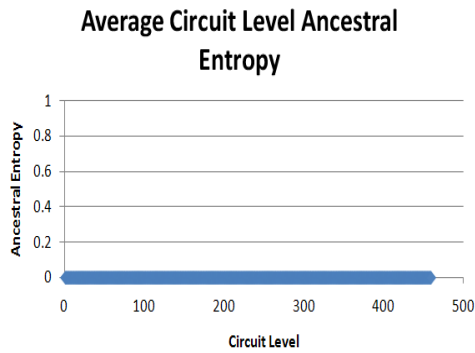
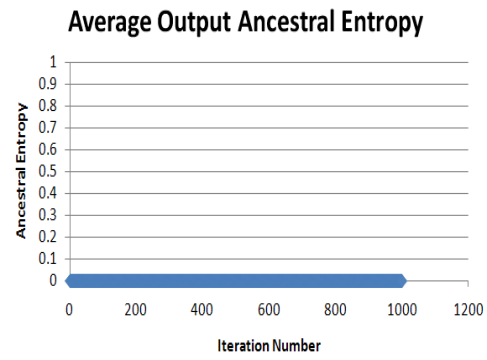


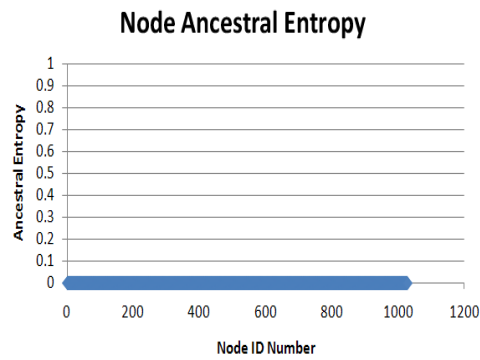
Figure A.11: OutputLevelTwoGates 11 Input C-17 Series Variant Circuits
(a),(b),(c) and (d)



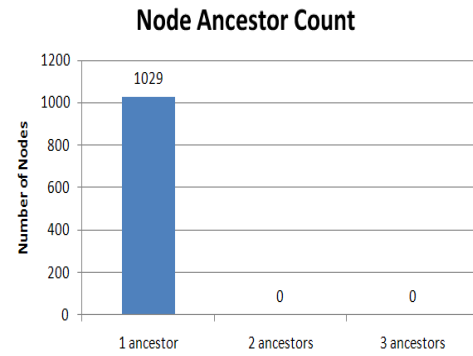
(a)



(b)



(c)



(d)

Figure A.12: OutputLevelTwoGates 11 Input C-17 Series - Iteration 1000
(a),(b),(c) and (d)

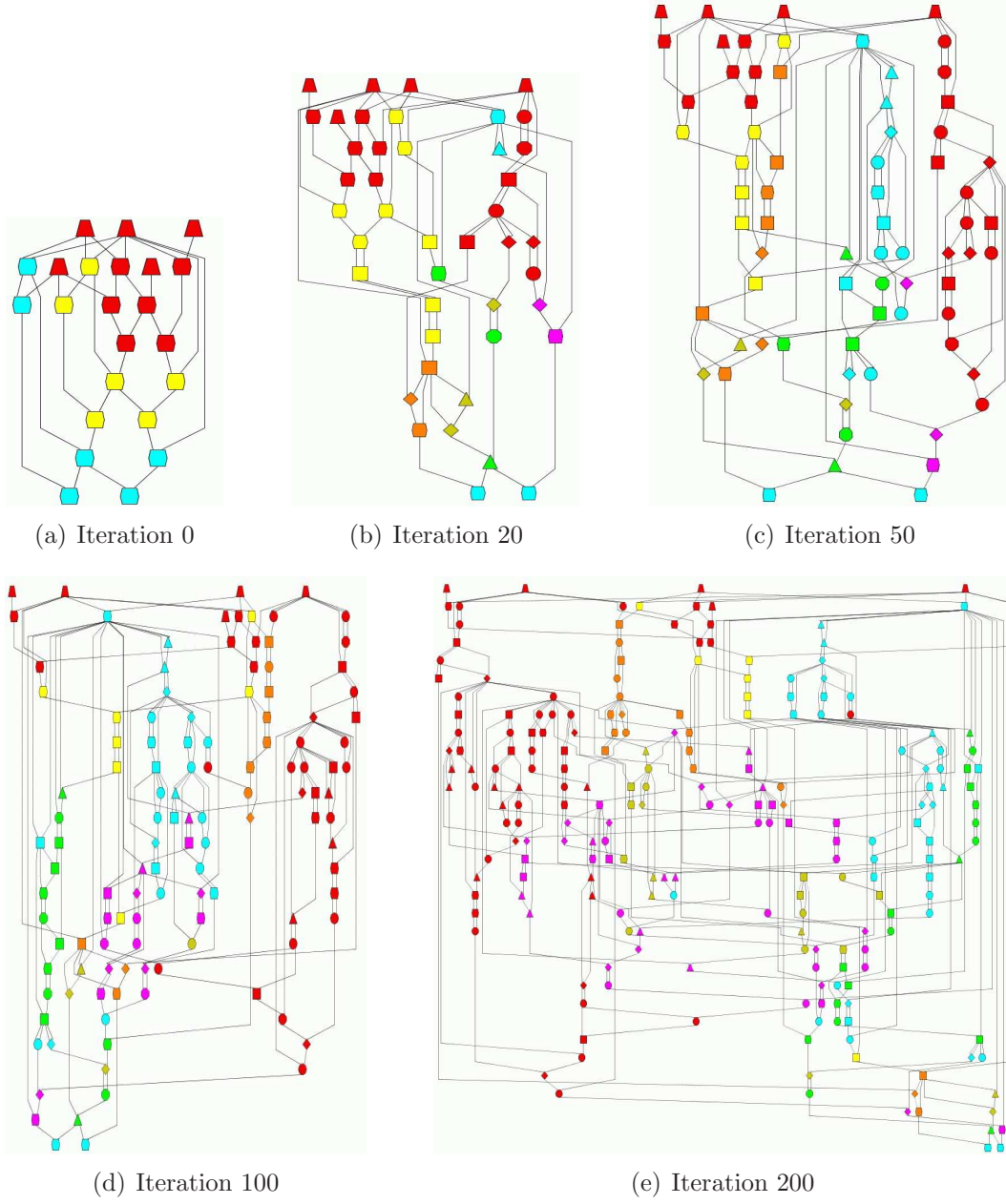
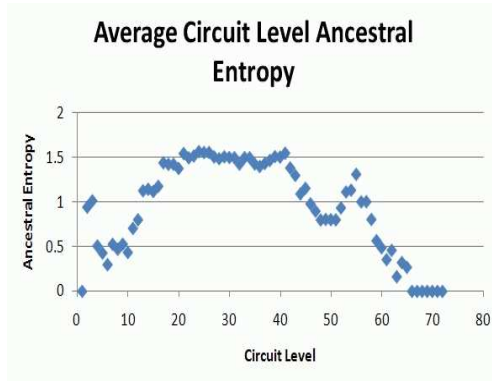
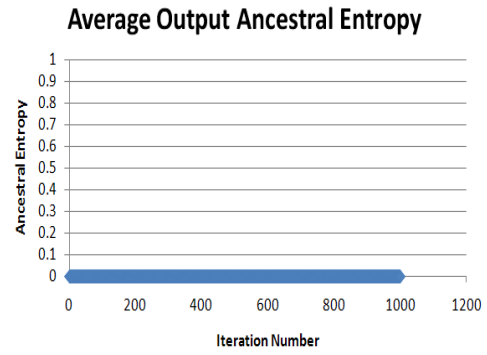


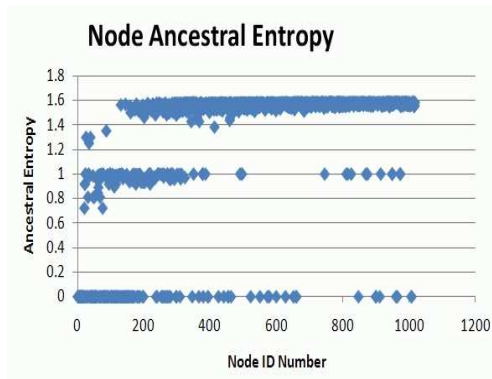
Figure A.13: LargestLevelTwoGates 5 Input C-17 Series Variant Circuits
(a),(b),(c), (d), and (e)



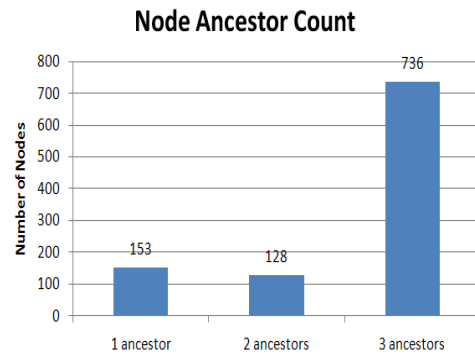
(a)



(b)



(c)



(d)

Figure A.14: LargestLevelTwoGates 5 Input C-17 Series - Iteration 1000
(a),(b),(c) and (d)

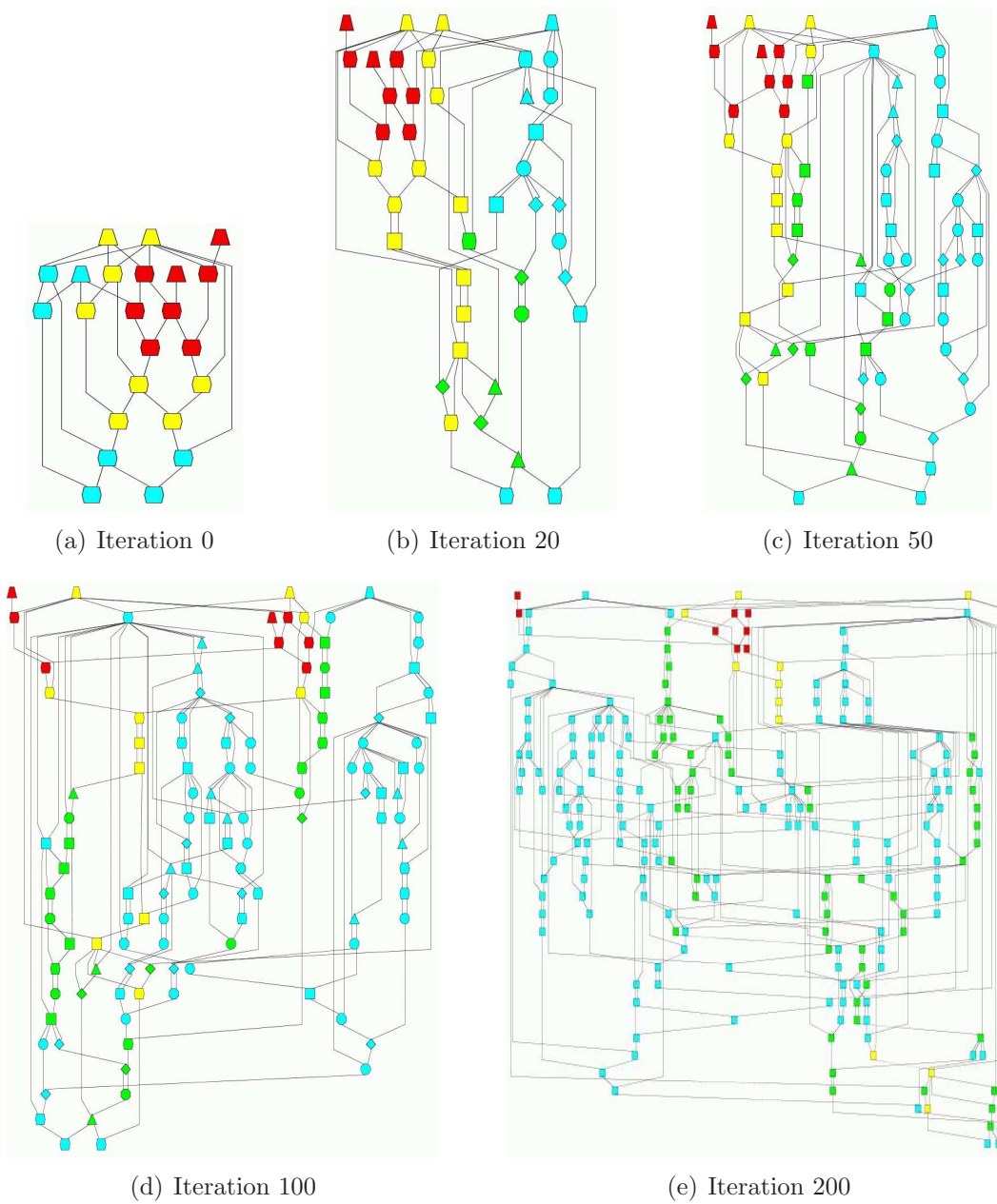
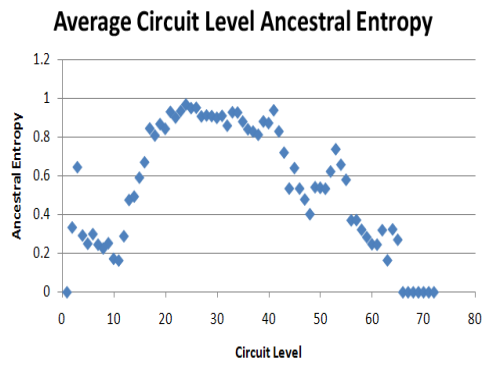
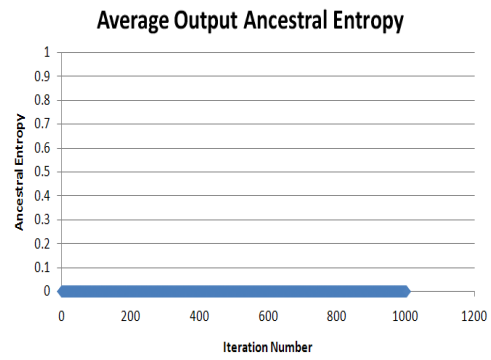


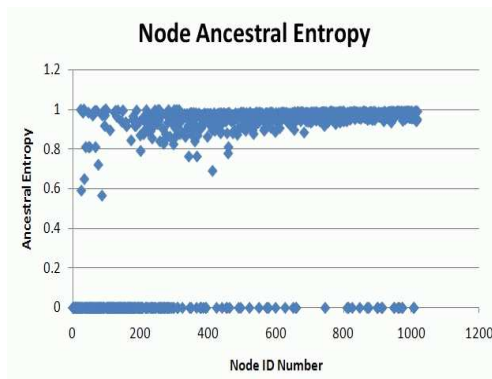
Figure A.15: LargestLevelTwoGates 5 ‘Split’ Input C-17 Series Variant Circuits
(a),(b),(c), (d), and (d)



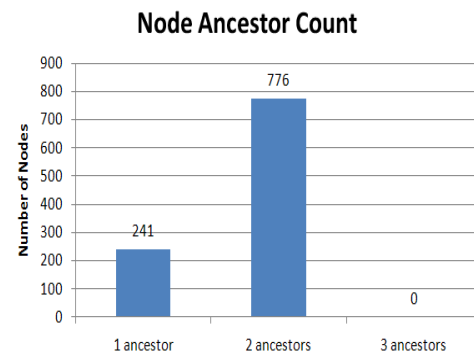
(a)



(b)



(c)



(d)

Figure A.16: LargestLevelTwoGates 5 ‘Split’ Input C-17 Series - Iteration 1000
(a),(b),(c), and (d)

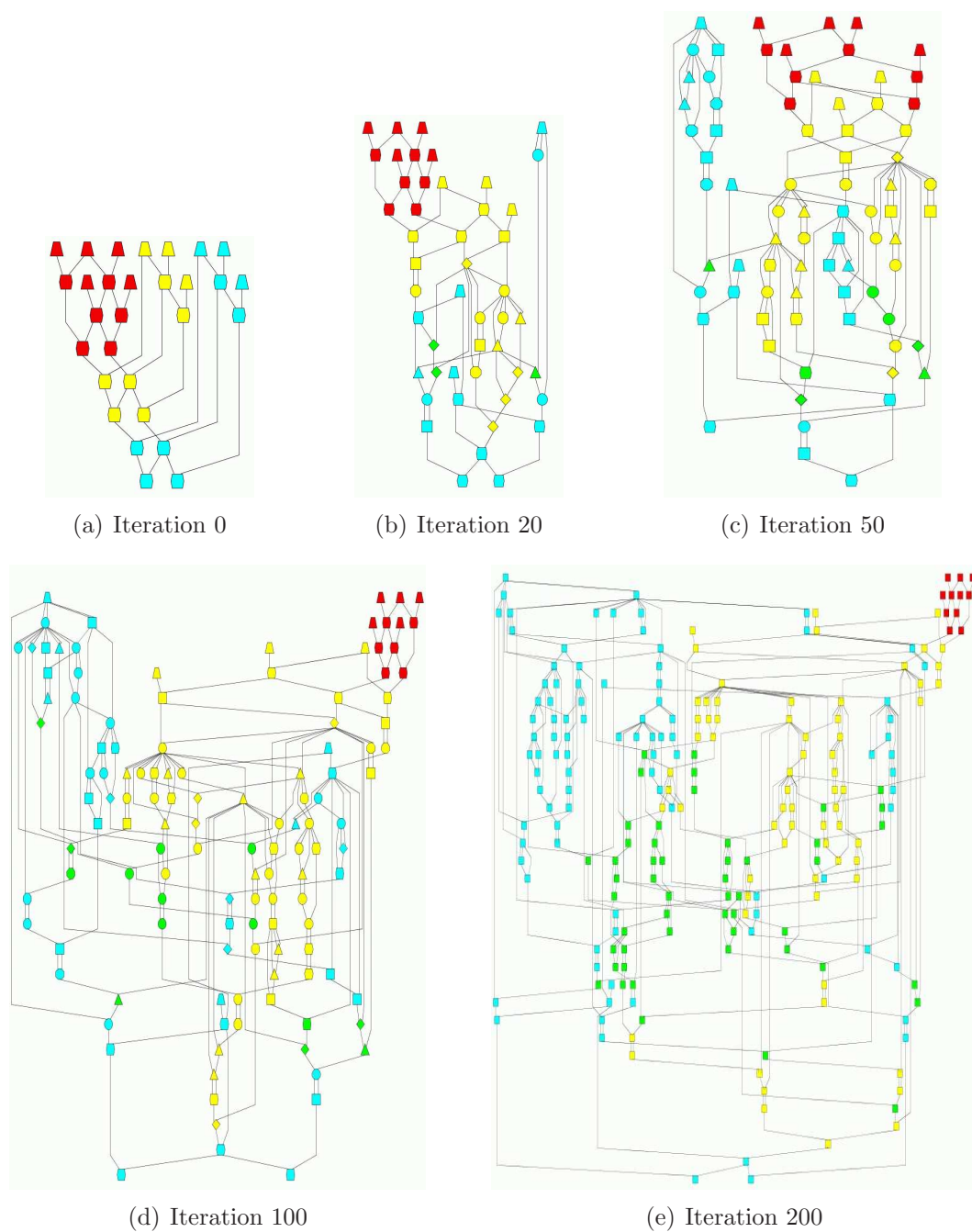
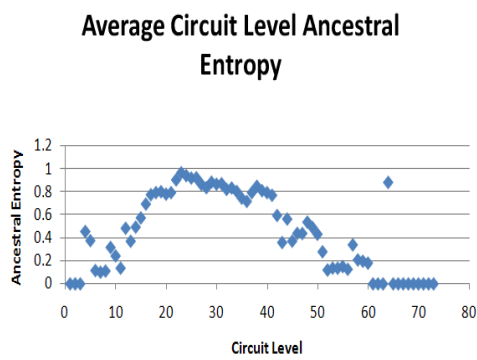
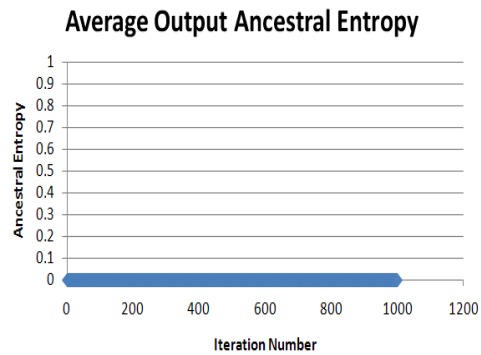


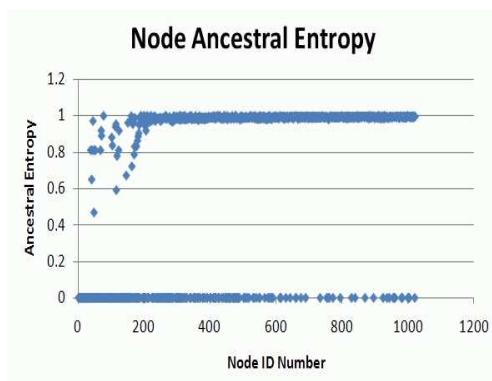
Figure A.17: LargestLevelTwoGates 11 Input C-17 Series Variant Circuits
(a),(b),(c), (d), and (e)



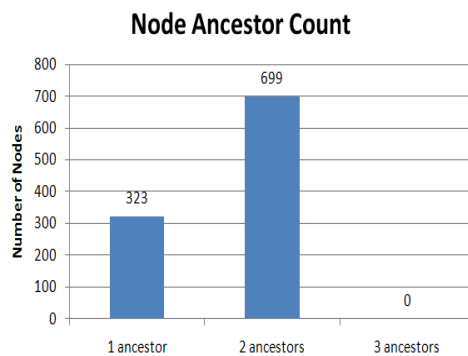
(a)



(b)



(c)



(d)

Figure A.18: LargestLevelTwoGates 11 Input C-17 Series - Iteration 1000
(a),(b),(c) and (d)

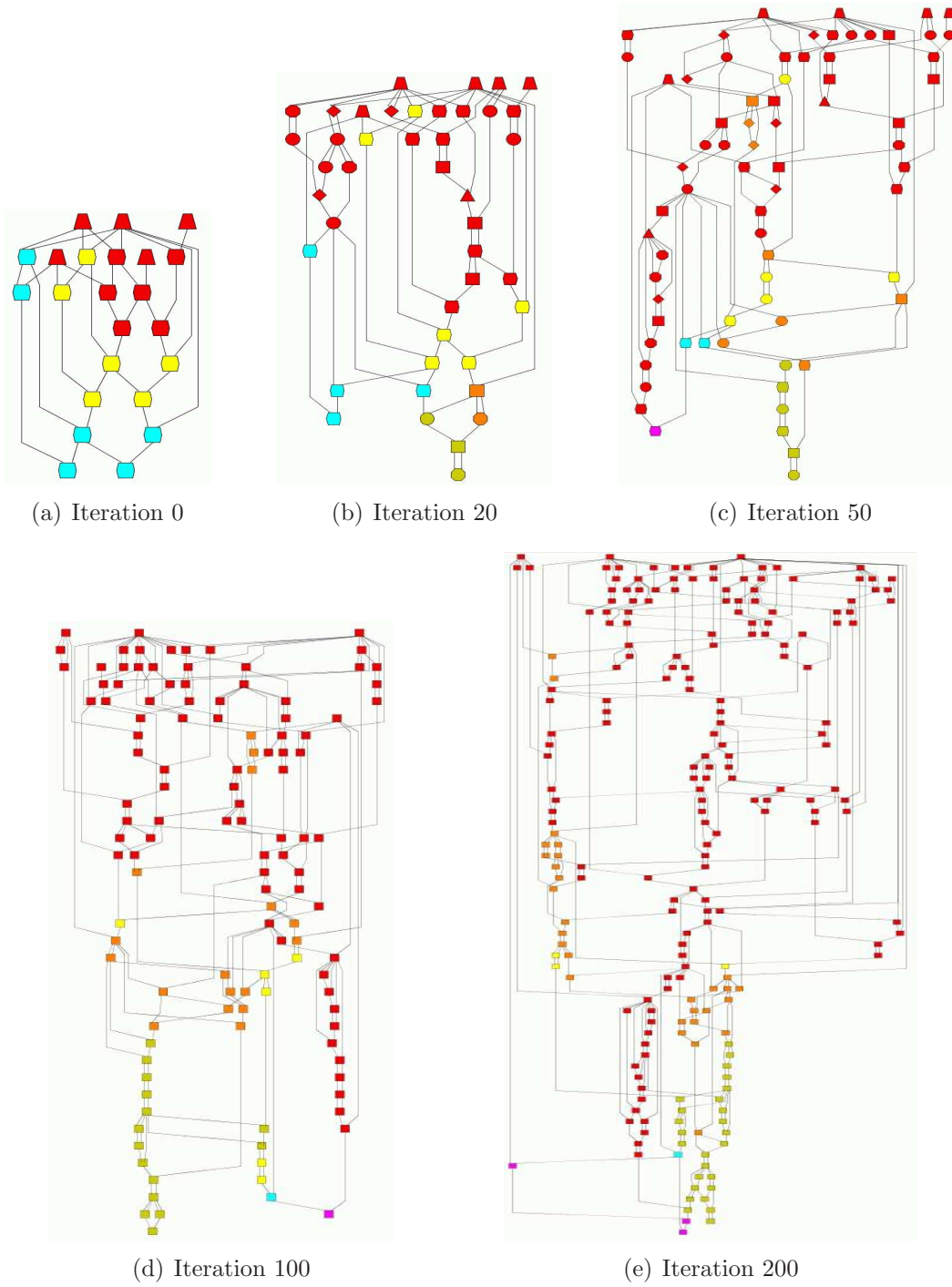
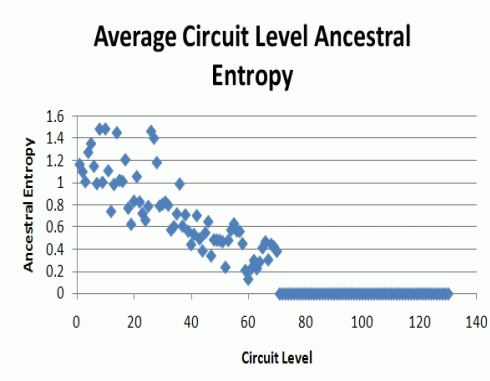
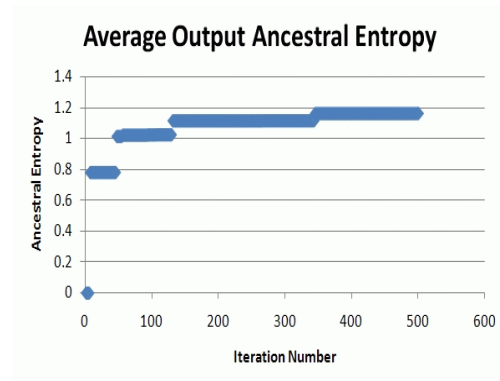


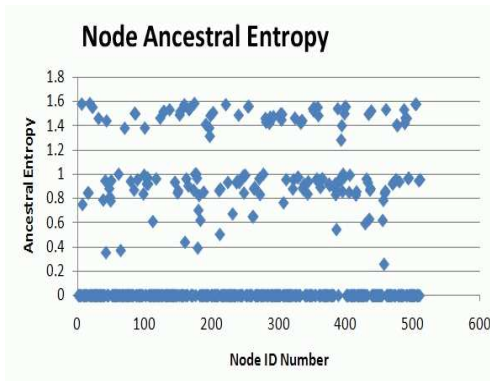
Figure A.19: RandomLevelTwoGates 5 Input C-17 Series Variant Circuits
(a),(b),(c), (d), and (e)



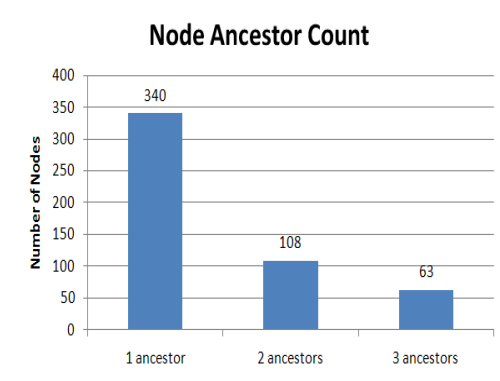
(a)



(b)



(c)



(d)

Figure A.20: RandomLevelTwoGates 5 Input C-17 Series - Iteration 500
(a),(b),(c) and (d)

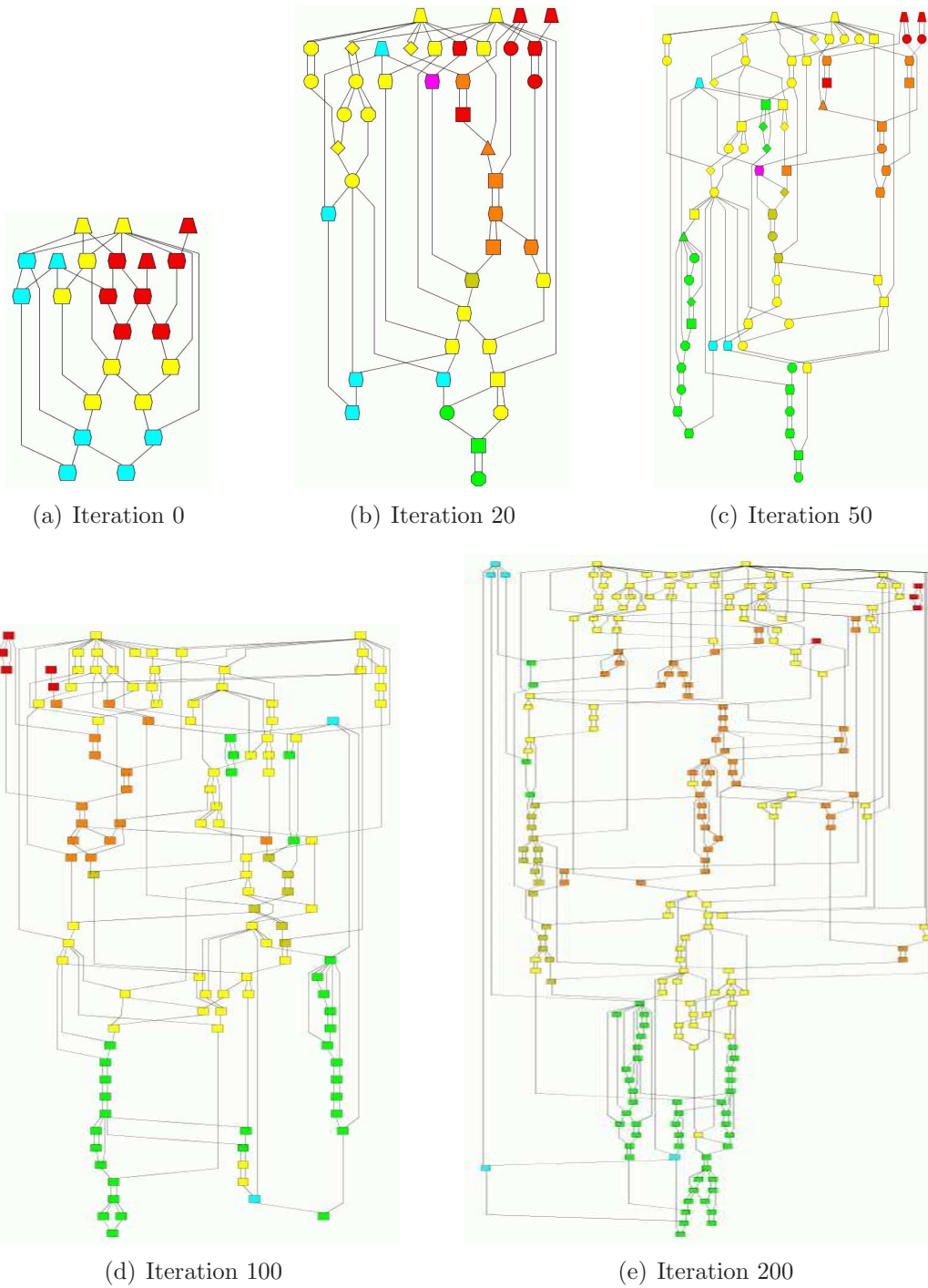
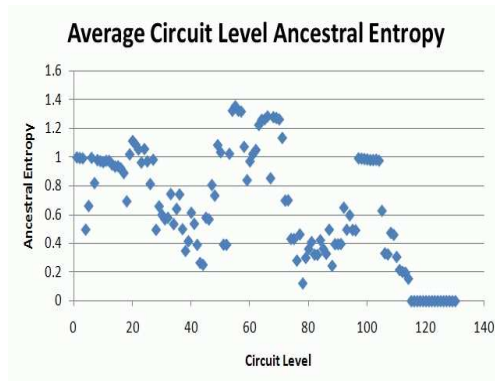
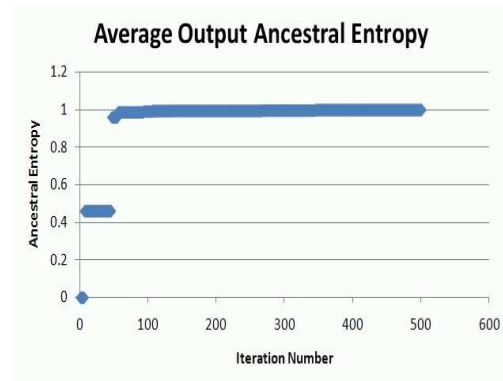


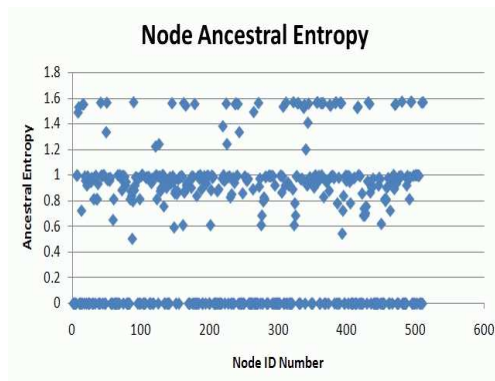
Figure A.21: RandomLevelTwoGates 5 ‘Split’ Input C-17 Series Variant Circuits
(a),(b),(c), (d), and (d)



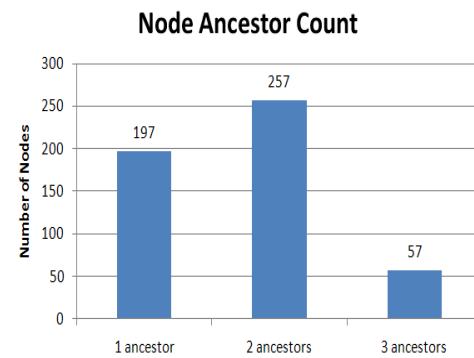
(a)



(b)



(c)



(d)

Figure A.22: RandomLevelTwoGates 5 ‘Split’ Input C-17 Series - Iteration 500
(a),(b),(c), and (d)

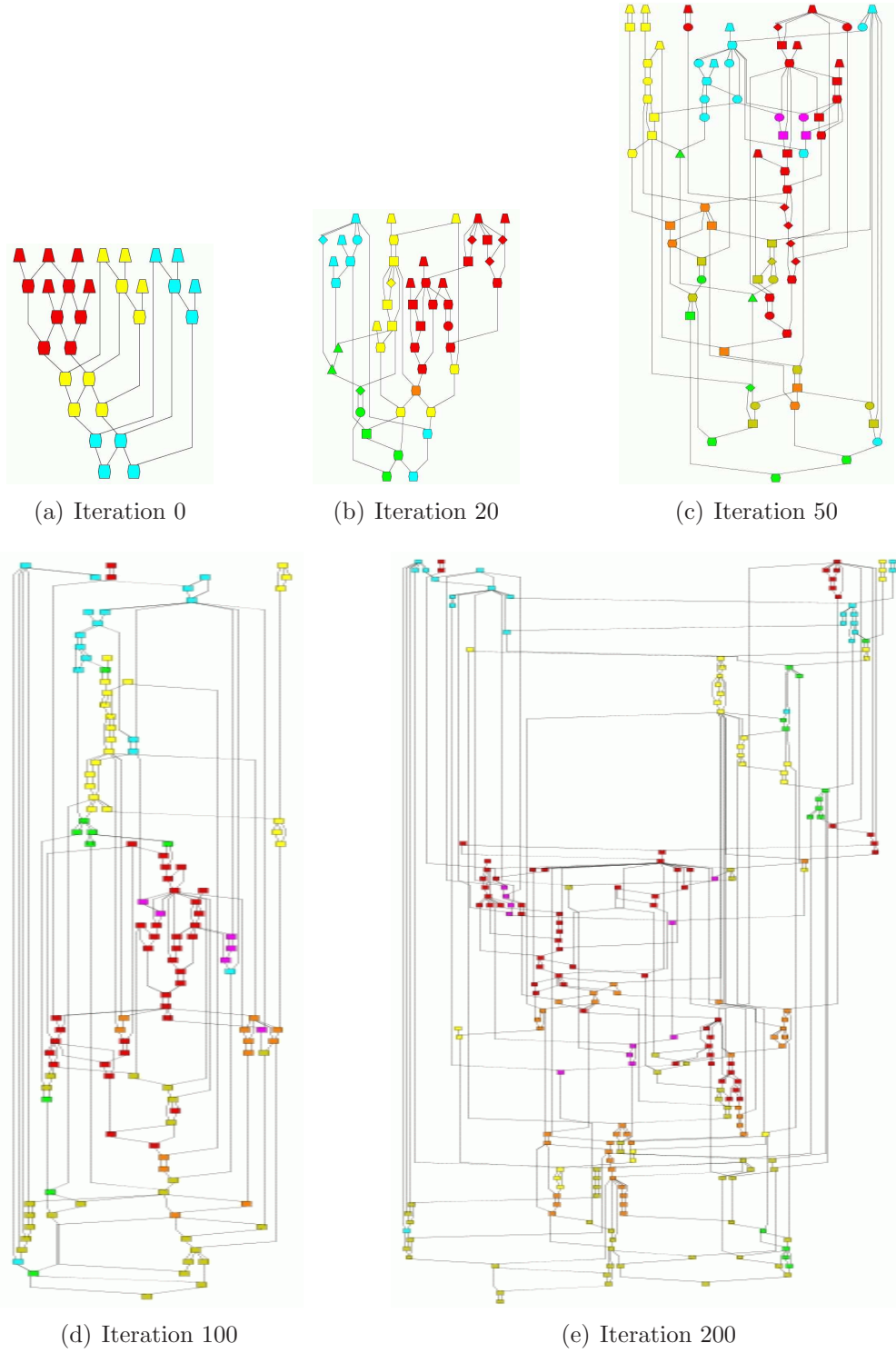
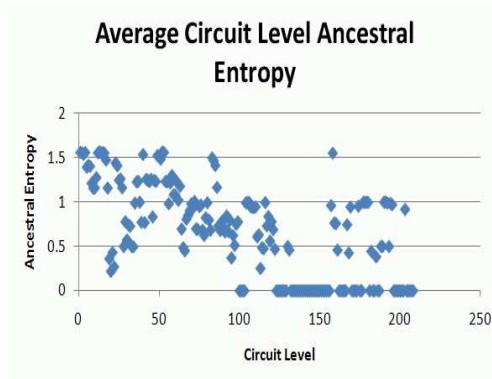
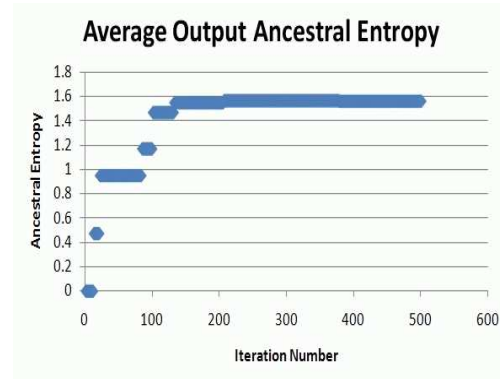


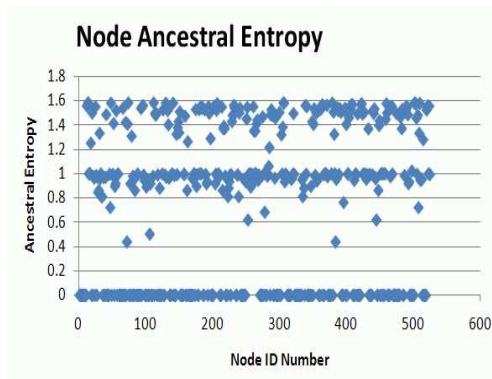
Figure A.23: RandomLevelTwoGates 11 Input C-17 Series Variant Circuits
(a),(b),(c), (d), and (e)



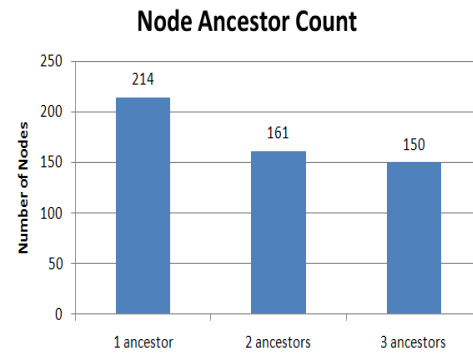
(a)



(b)



(c)



(d)

Figure A.24: RandomLevelTwoGates 11 Input C-17 Series - Iteration 500
(a),(b),(c) and (d)

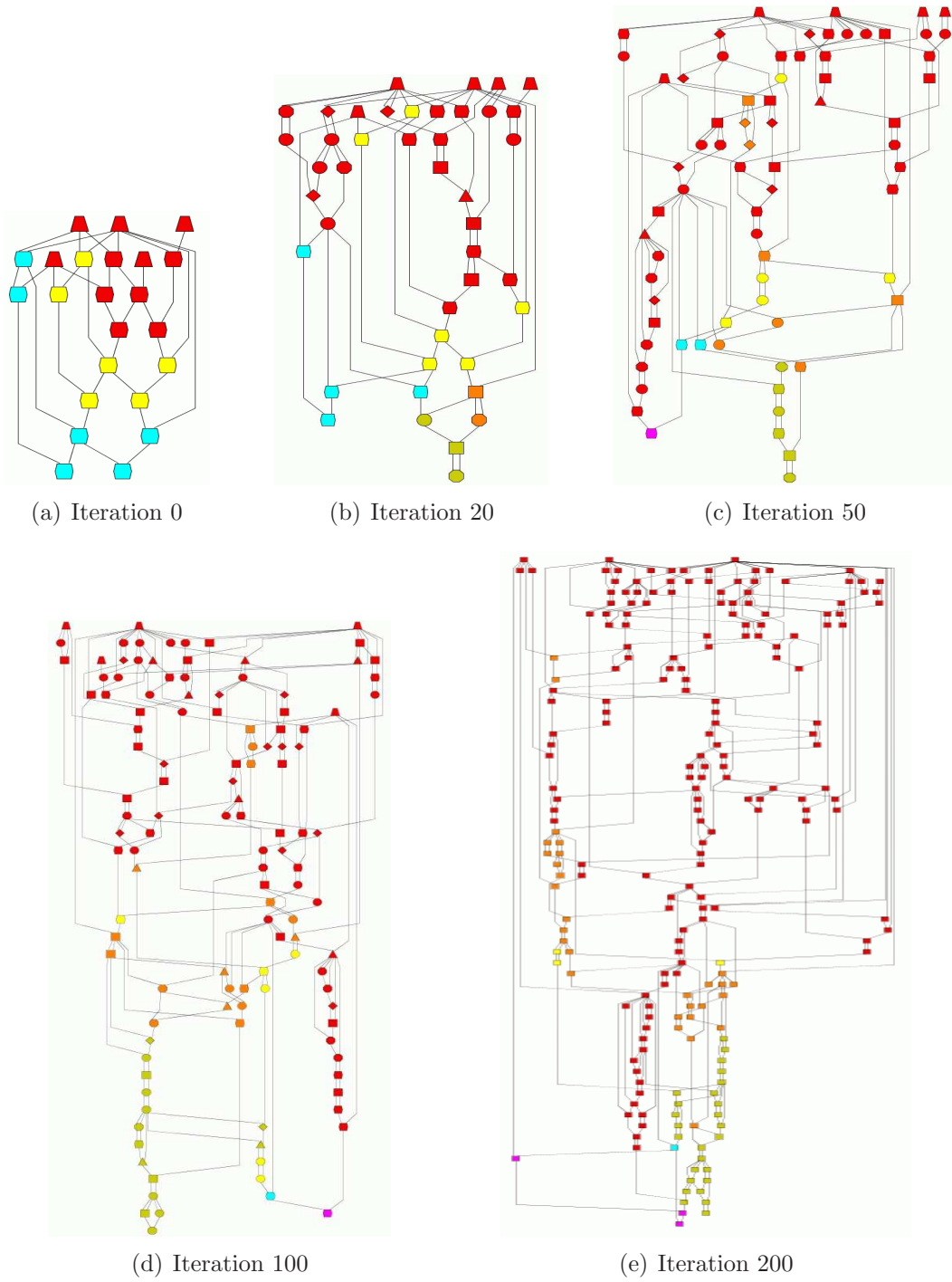
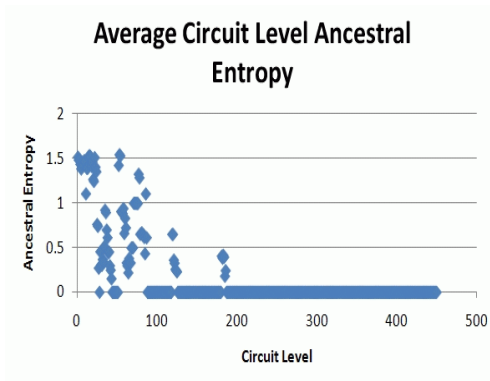
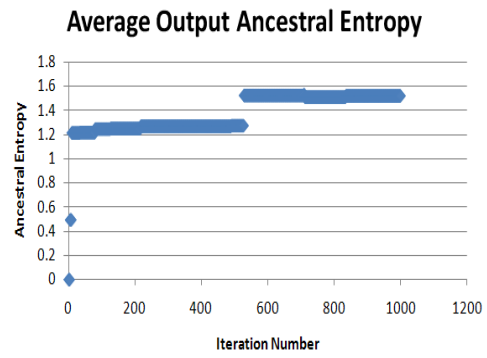


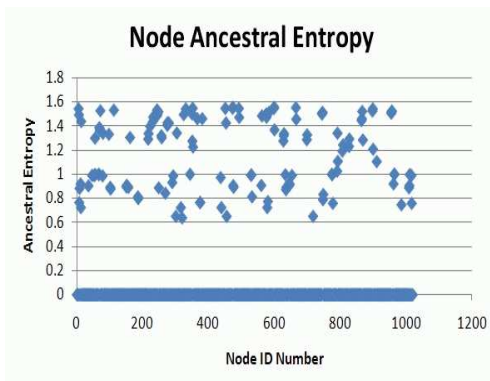
Figure A.25: RandomTwoGates 5 Input C-17 Series Variant Circuits
(a),(b),(c), (d), and (e)



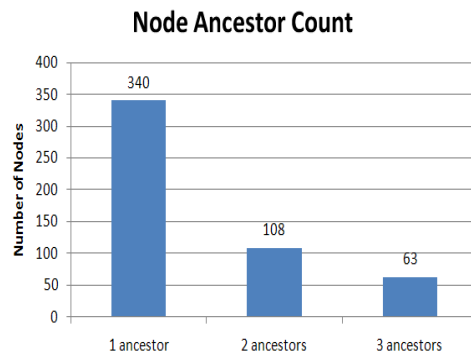
(a)



(b)



(c)



(d)

Figure A.26: RandomTwoGates 5 Input C-17 Series - Iteration 1000
(a),(b),(c) and (d)

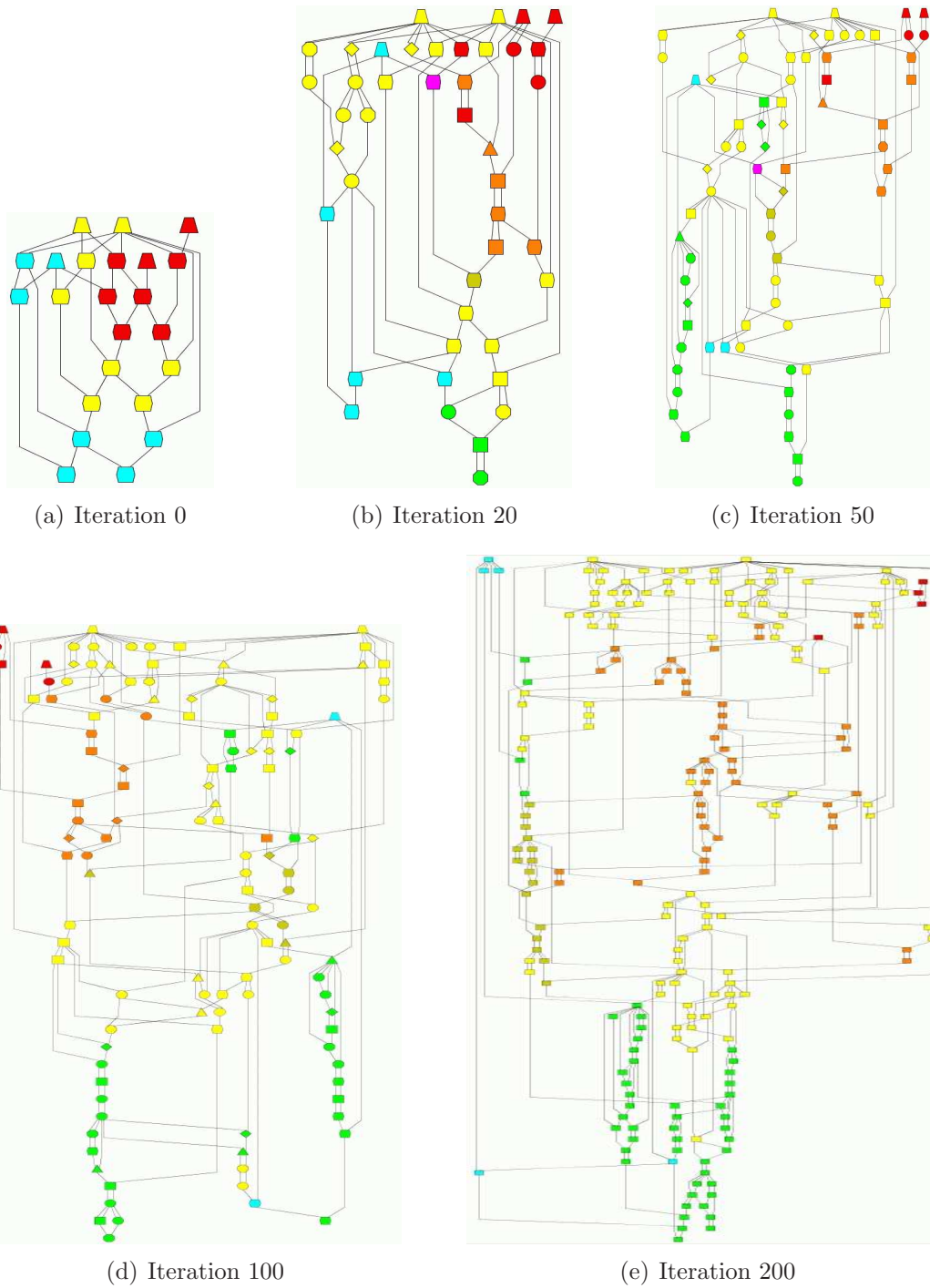
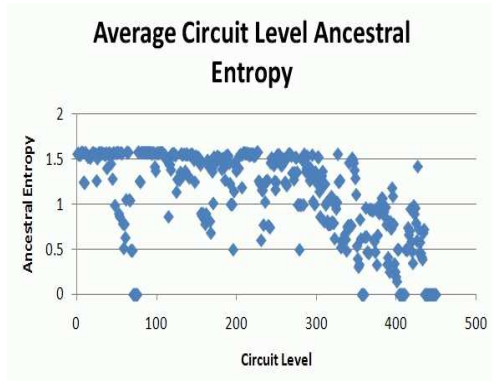
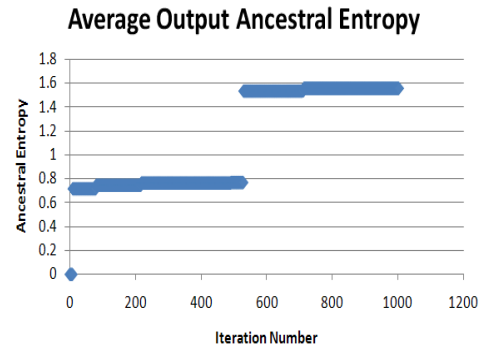


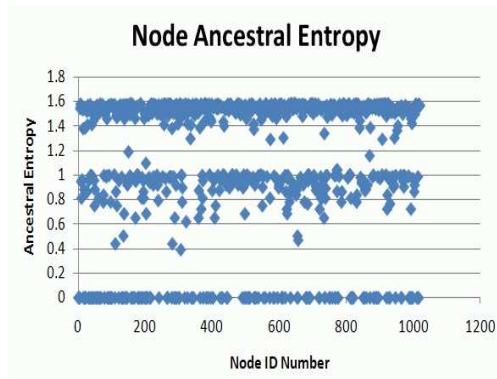
Figure A.27: RandomTwoGates 5 ‘Split’ Input C-17 Series Variant Circuits
(a),(b),(c), (d), and (d)



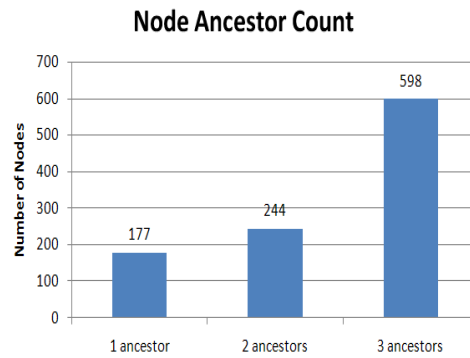
(a)



(b)



(c)



(d)

Figure A.28: RandomTwoGates 5 ‘Split’ Input C-17 Series - Iteration 1000
(a),(b),(c), and (d)

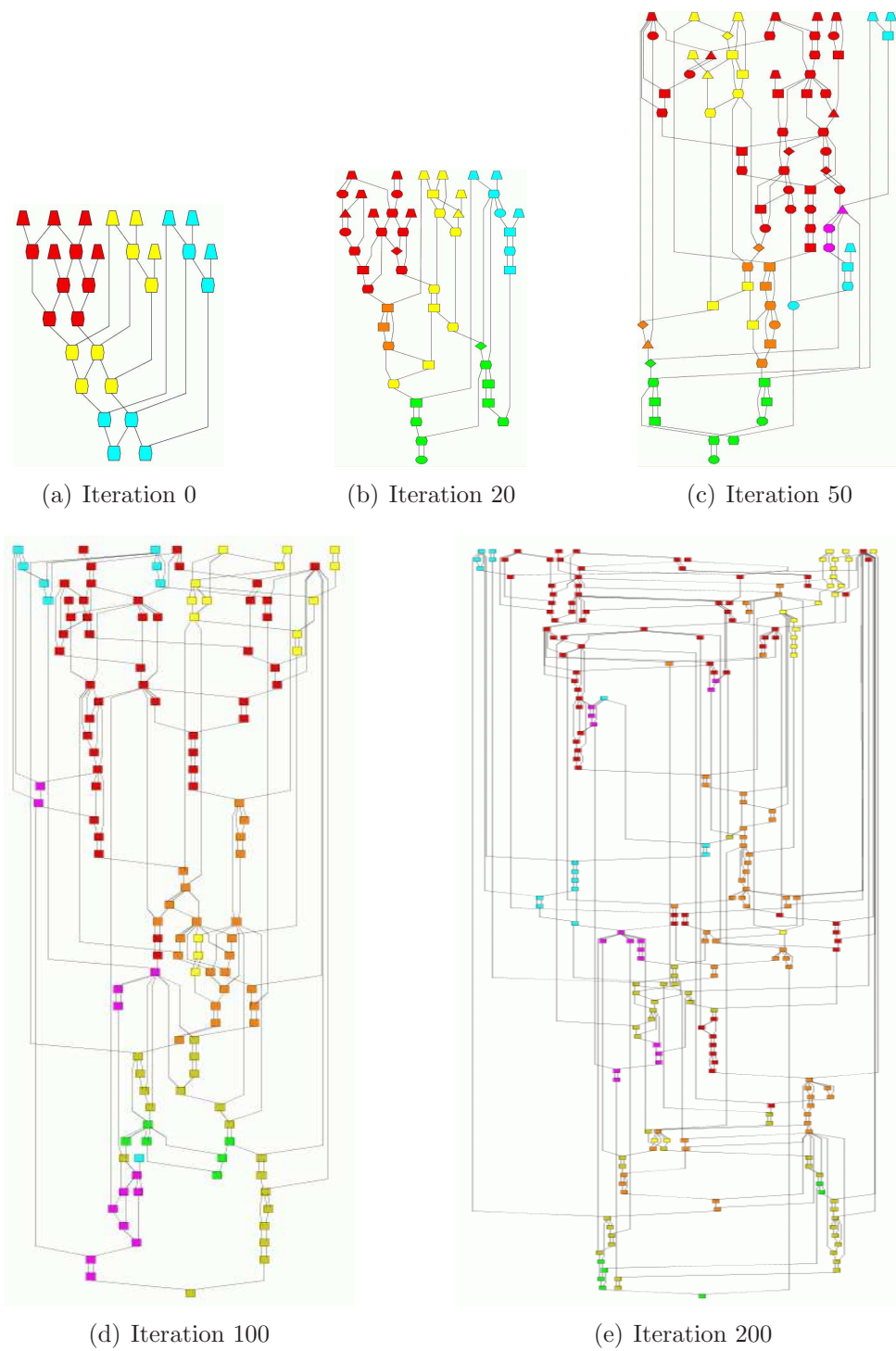
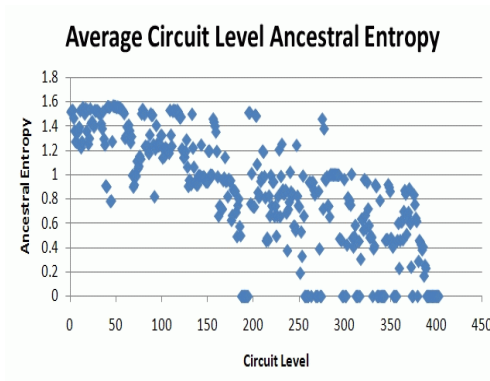
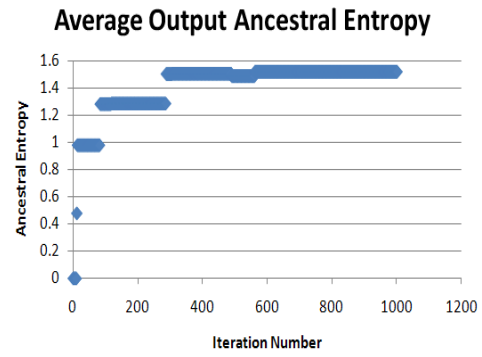


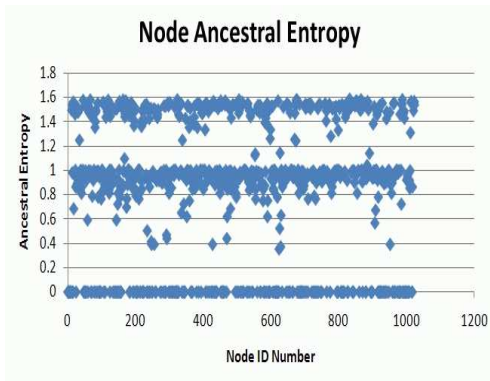
Figure A.29: RandomTwoGates 11 Input C-17 Series Variant Circuits
(a),(b),(c), (d), and (e)



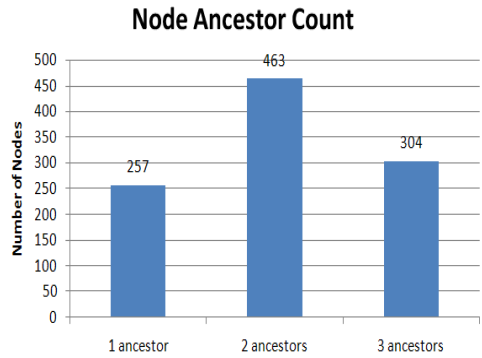
(a)



(b)

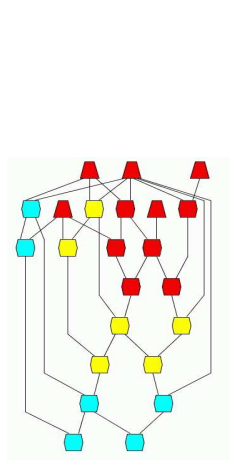


(c)

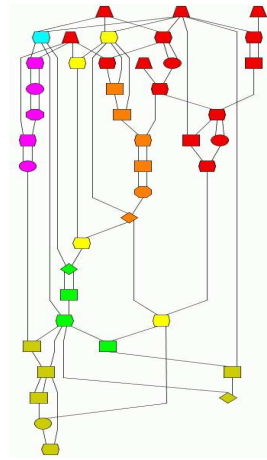


(d)

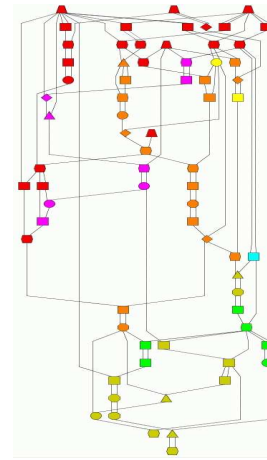
Figure A.30: RandomTwoGates 11 Input C-17 Series - Iteration 1000
(a),(b),(c) and (d)



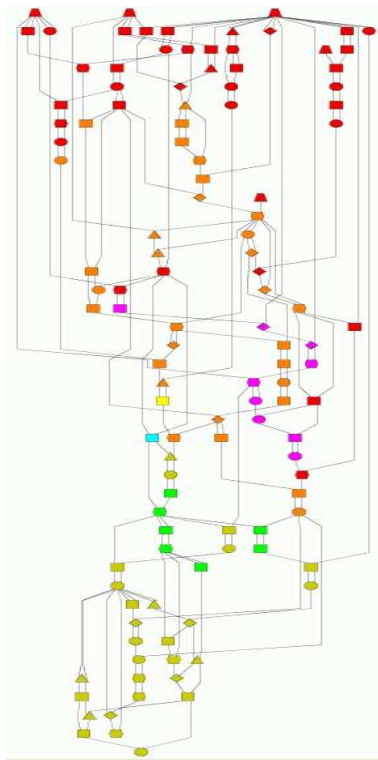
(a) Iteration 0



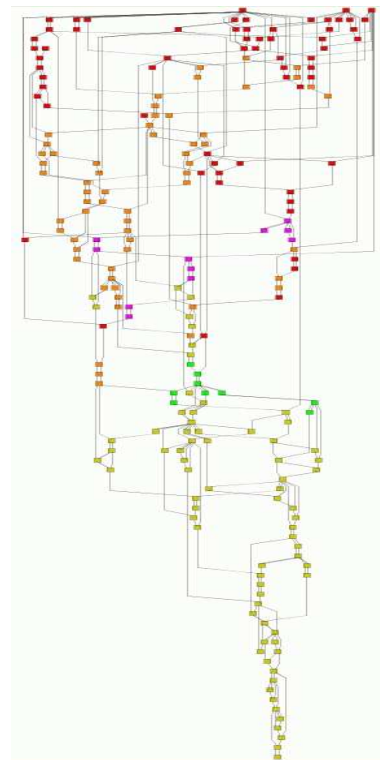
(b) Iteration 20



(c) Iteration 50

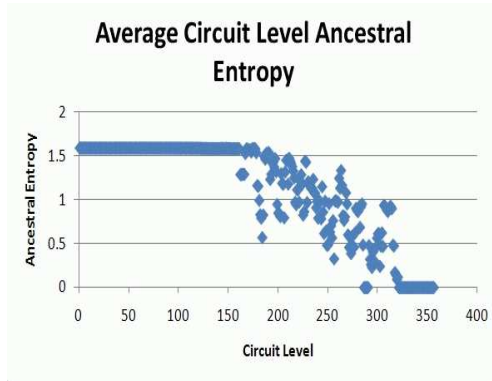


(d) Iteration 100

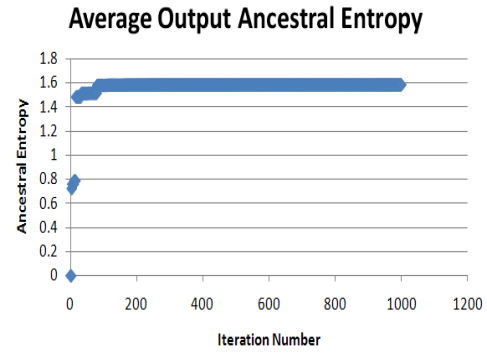


(e) Iteration 200

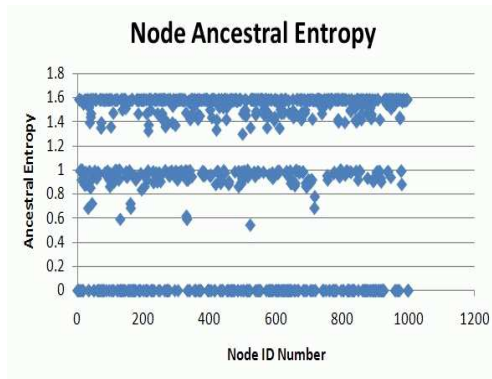
Figure A.31: RandomAlgorithm 5 Input C-17 Series Variant Circuits
(a),(b),(c), (d), and (e)



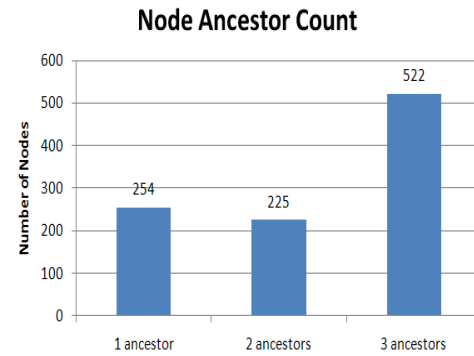
(a)



(b)



(c)



(d)

Figure A.32: Random Algorithm 5 Input C-17 Series - Iteration 1000
(a),(b),(c) and (d)

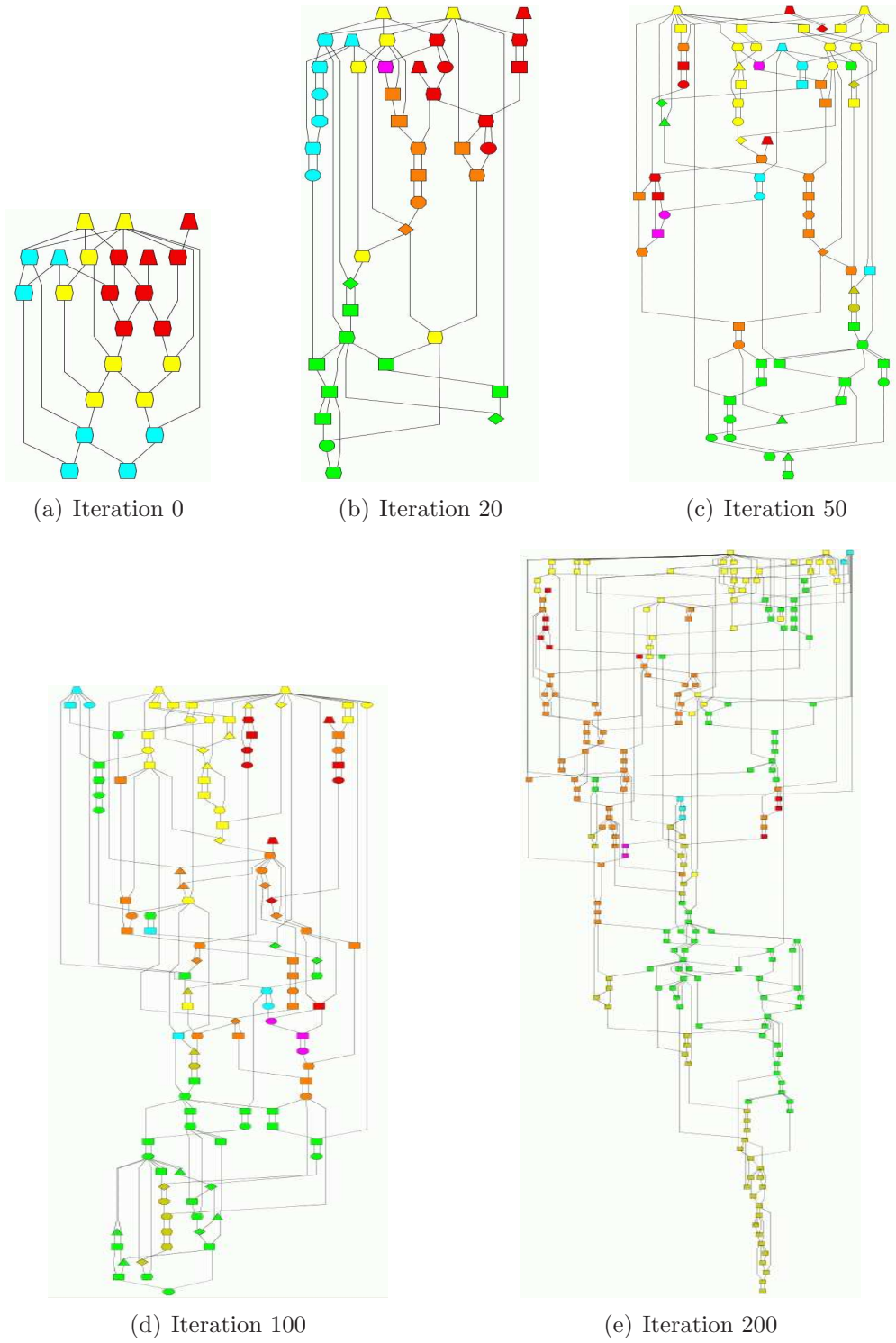
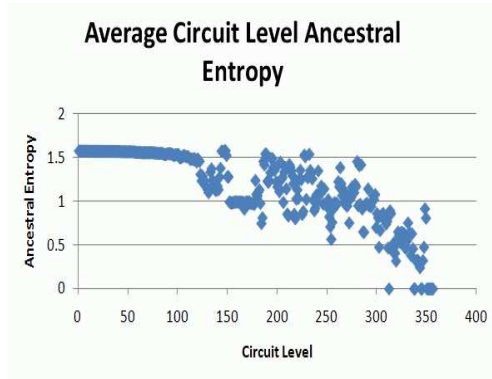
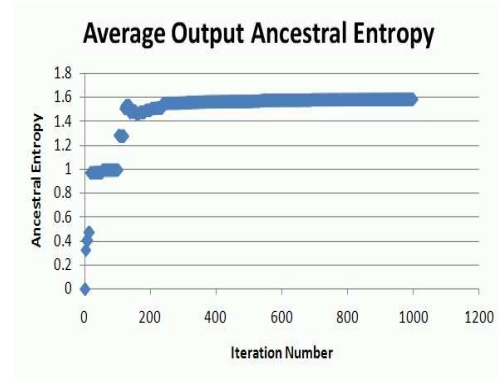


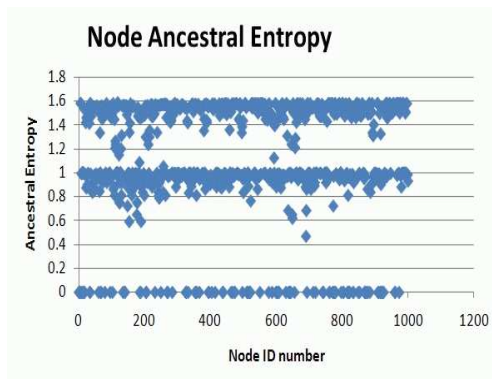
Figure A.33: RandomAlgorithm 5 ‘Split’ Input C-17 Series Variant Circuits
(a),(b),(c), (d), and (d)



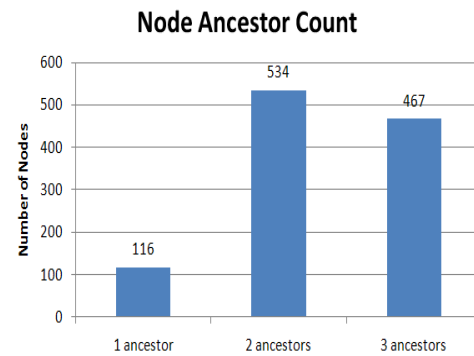
(a)



(b)



(c)



(d)

Figure A.34: RandomAlgorithm 5 ‘Split’ Input C-17 Series - Iteration 500
(a),(b),(c), and (d)

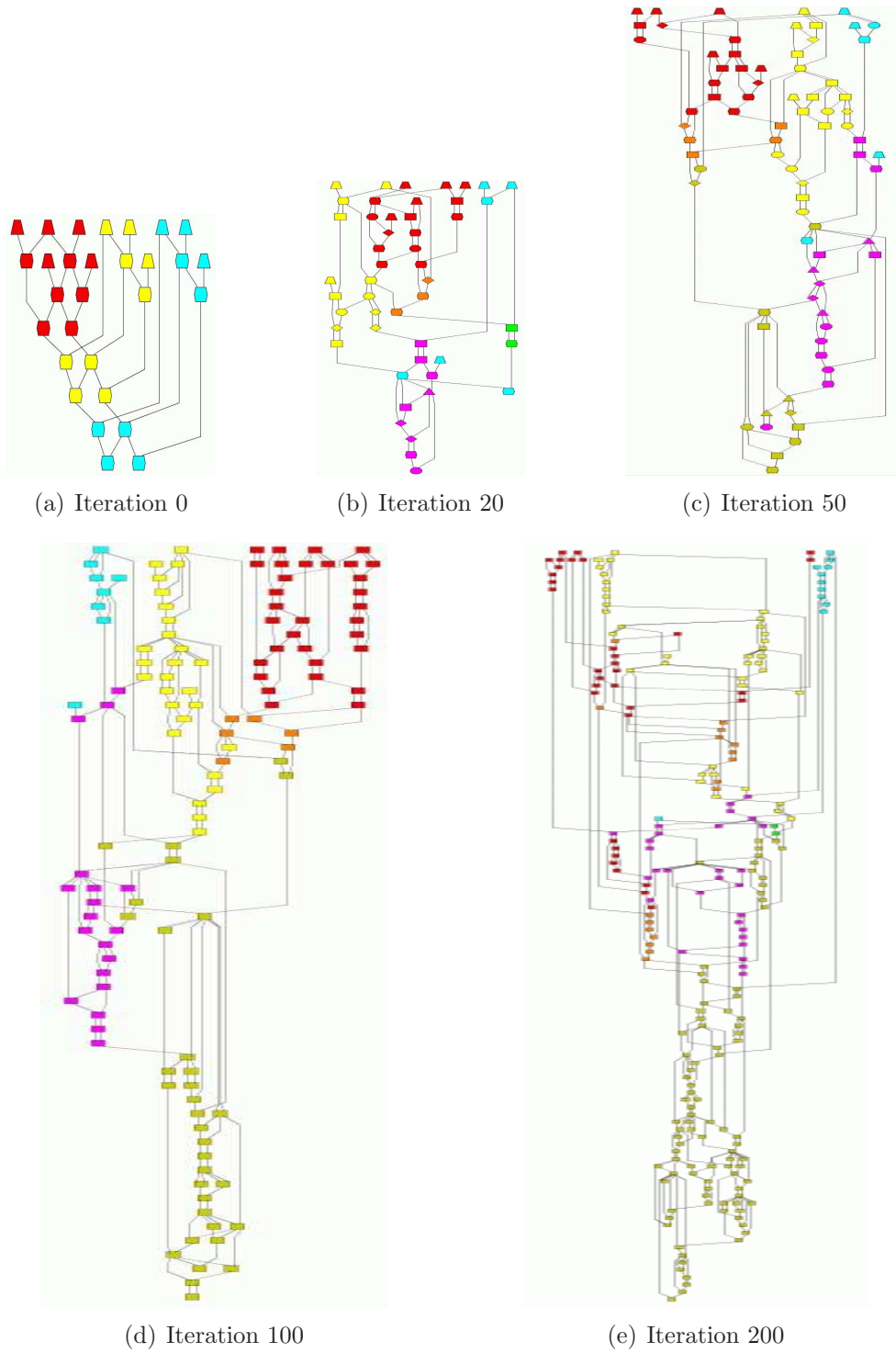
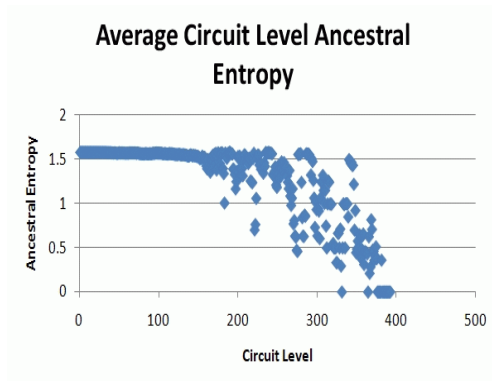
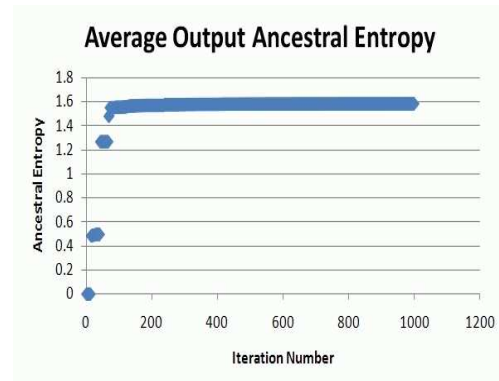


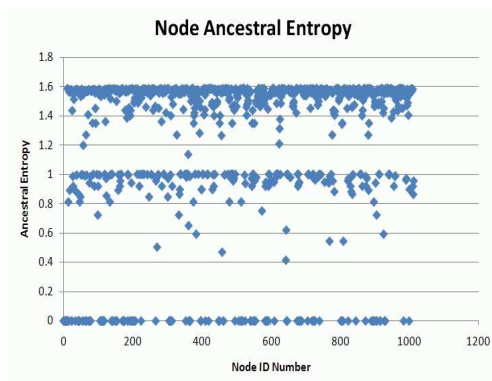
Figure A.35: RandomAlgorithm 11 Input C-17 Series Variant Circuits
(a),(b),(c), (d), and (e)



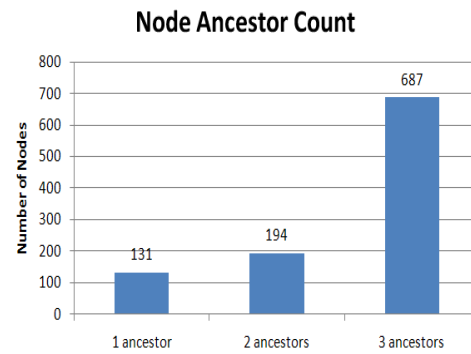
(a)



(b)



(c)



(d)

Figure A.36: RandomAlgorithm 11 Input C-17 Series - Iteration 1000
(a),(b),(c) and (d)

Appendix B. Three Gate Replacement Parallel Circuit Variant

Graphs and Charts

This appendix contains the graphs and charts for the parallel circuits discussed in Chapter IV. These charts and graphs are from one experiment only and are provided to show trends of each algorithm.

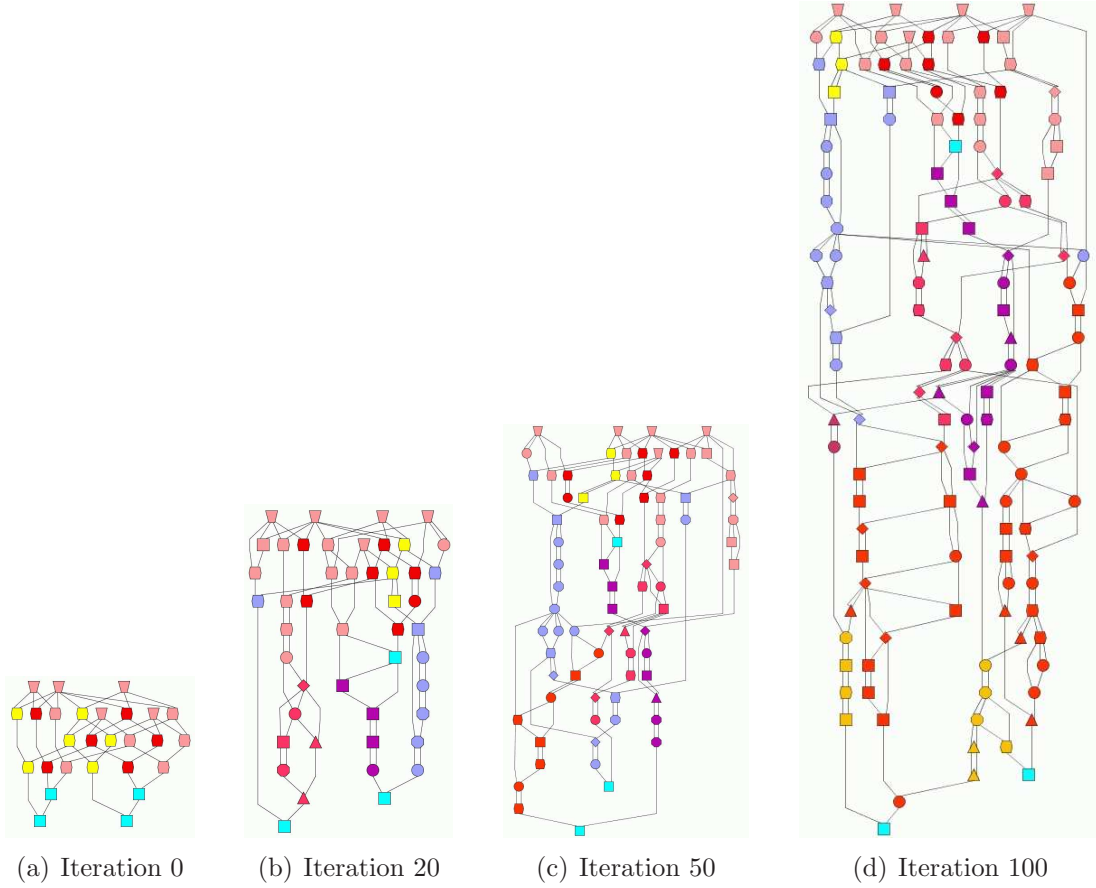
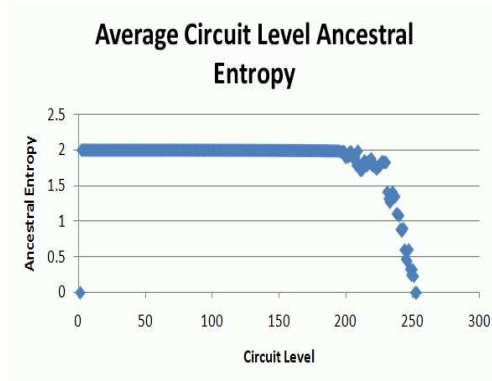
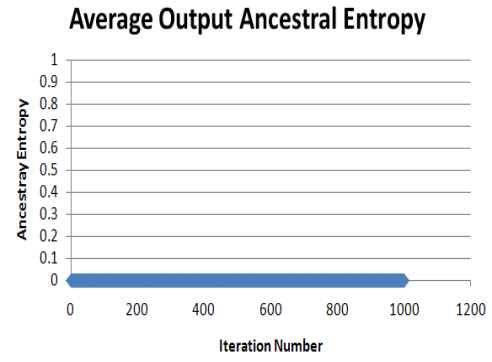


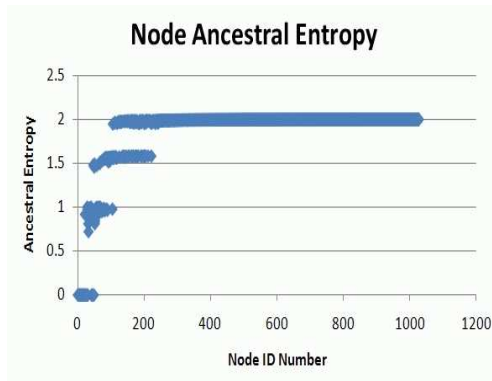
Figure B.1: FixedLevelTwoGates Shared Input C-17 Parallel Variant Circuits (a),(b),(c) and (d)



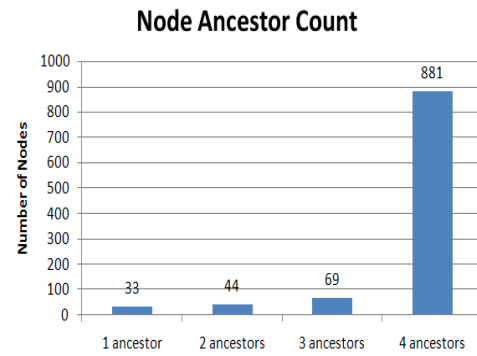
(a)



(b)



(c)



(d)

Figure B.2: FixedLevelTwoGates Shared Input C-17 Parallel - Iteration 1000
(a),(b),(c) and (d)

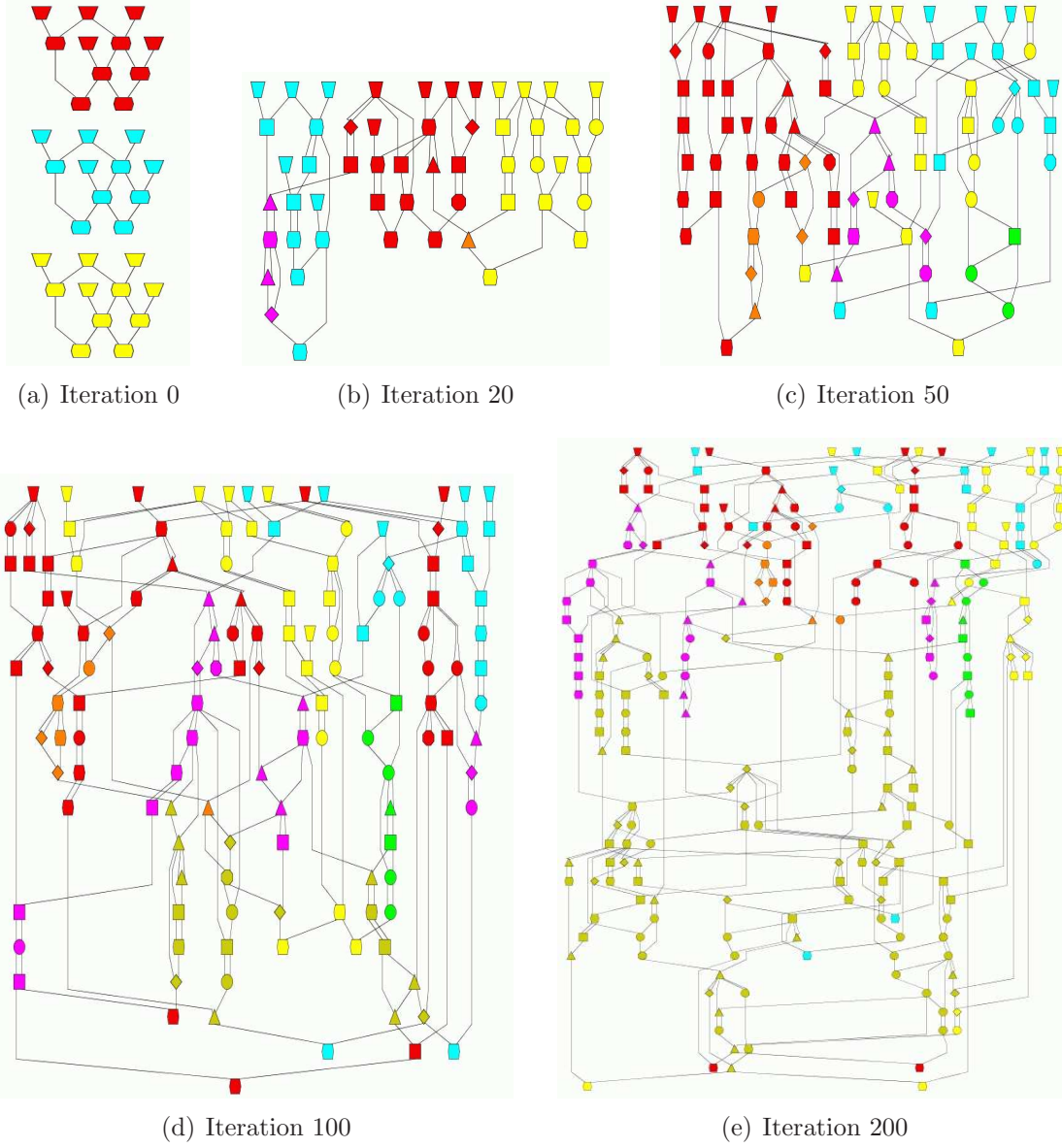
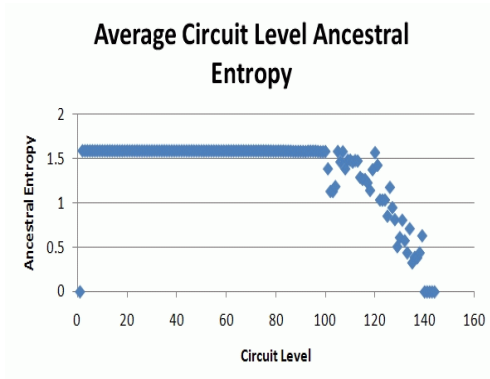
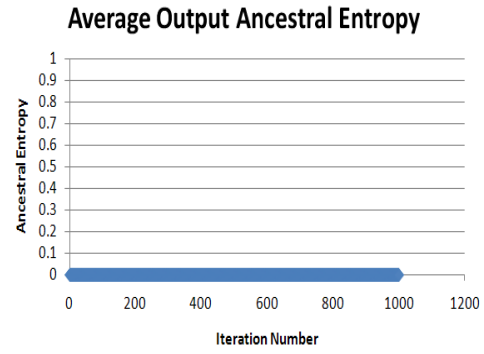


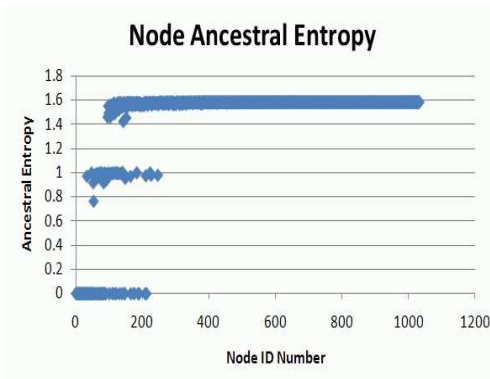
Figure B.3: FixedLevelTwoGates Individual Input C-17 Parallel Variant Circuits
(a),(b),(c),(d) and (e)



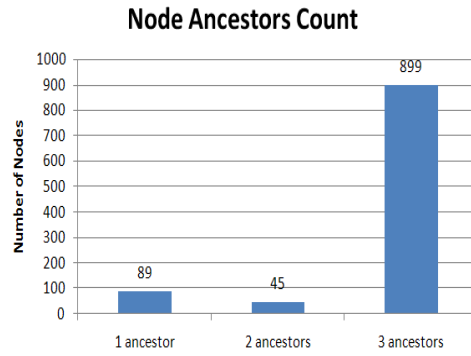
(a)



(b)



(c)



(d)

Figure B.4: FixedLevelTwoGates Individual Input C-17 Parallel - Iteration 1000
(a),(b),(c) and (d)

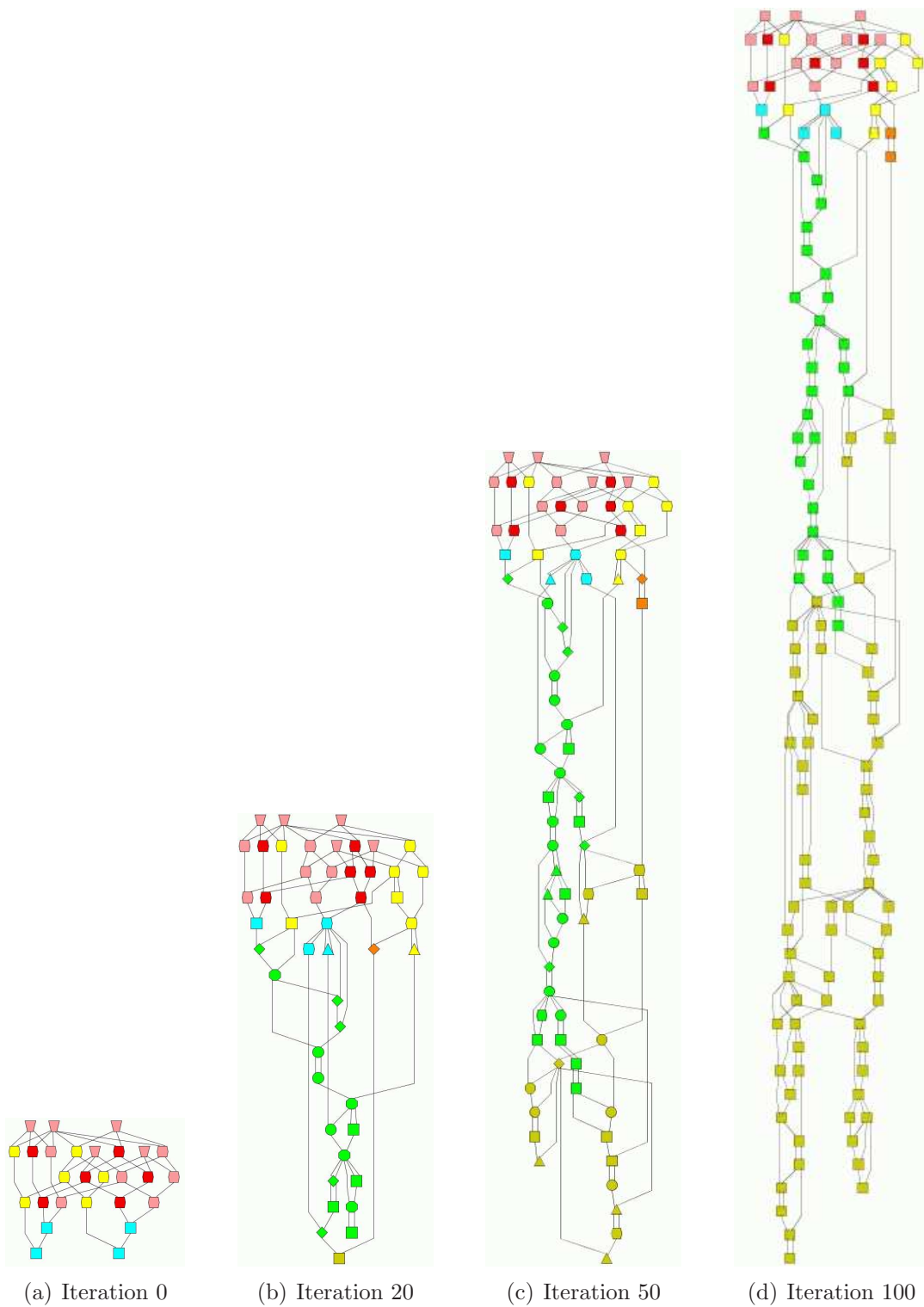
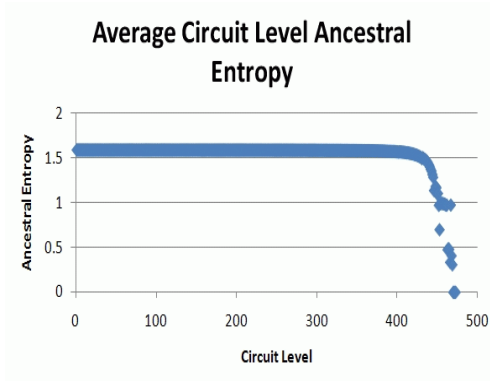
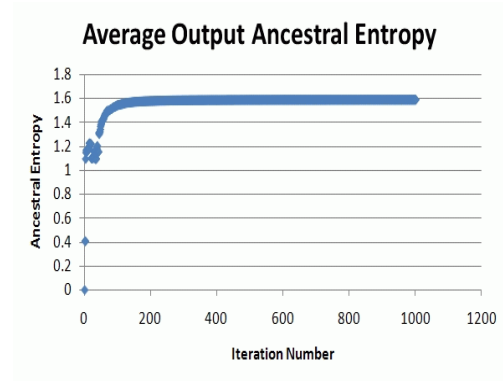


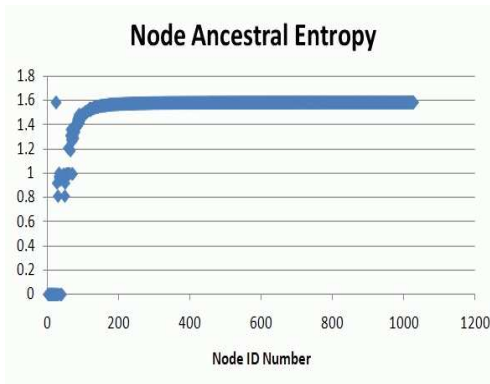
Figure B.5: OutputLevelTwoGates Shared Input C-17 Parallel Variant Circuits
(a),(b),(c) and (d)



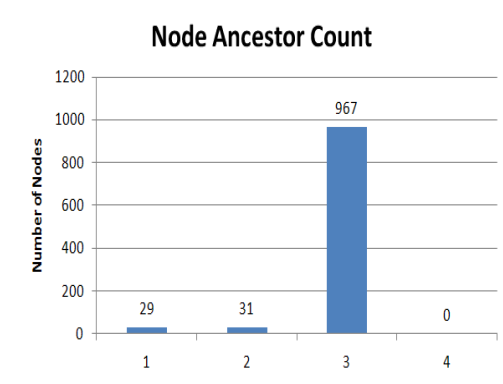
(a)



(b)



(c)



(d)

Figure B.6: OutputLevelTwoGates Shared Input C-17 Parallel - Iteration 1000
(a),(b),(c) and (d)

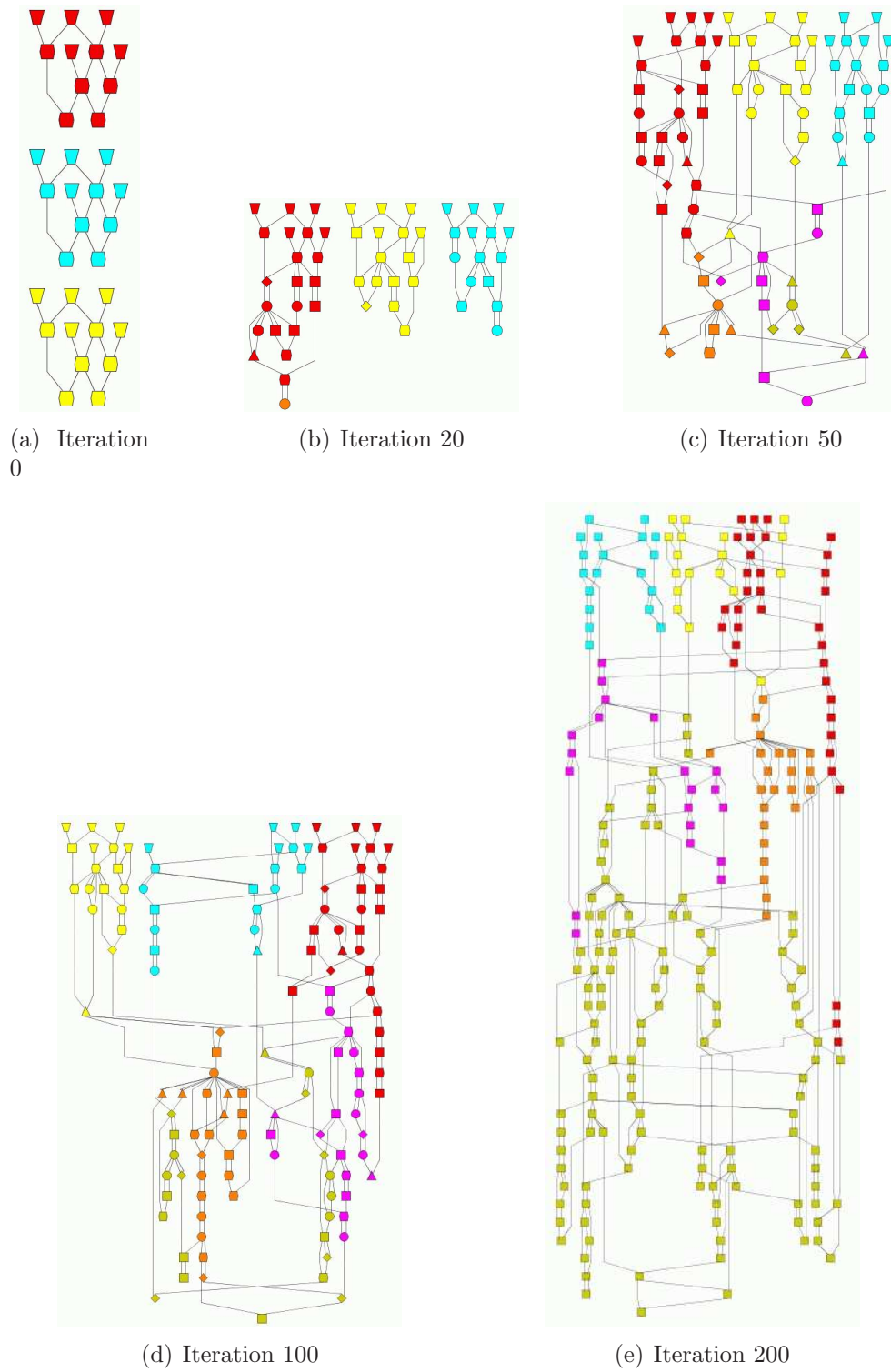
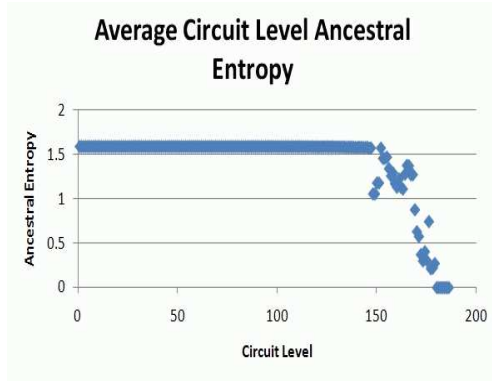
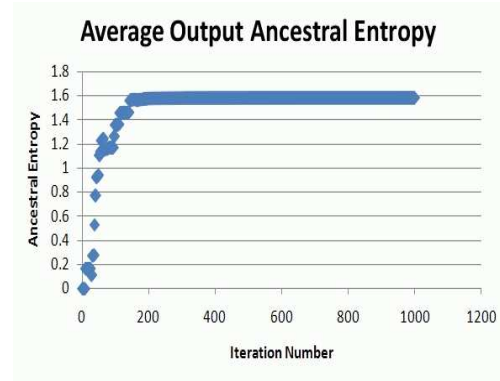


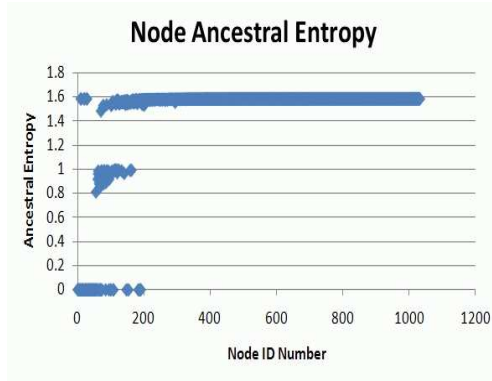
Figure B.7: OutputLevelTwoGates Individual Input C-17 Parallel Variant Circuits
(a),(b),(c), (d) and (e)



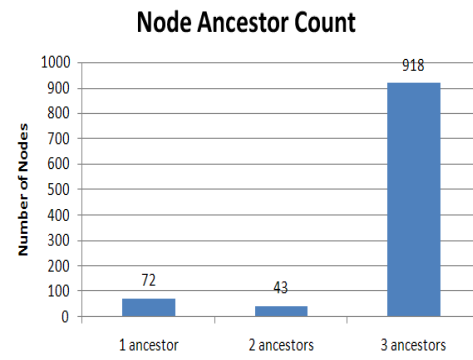
(a)



(b)



(c)



(d)

Figure B.8: OutputLevelTwoGates Individual Input C-17 Parallel - Iteration 1000
(a),(b),(c) and (d)

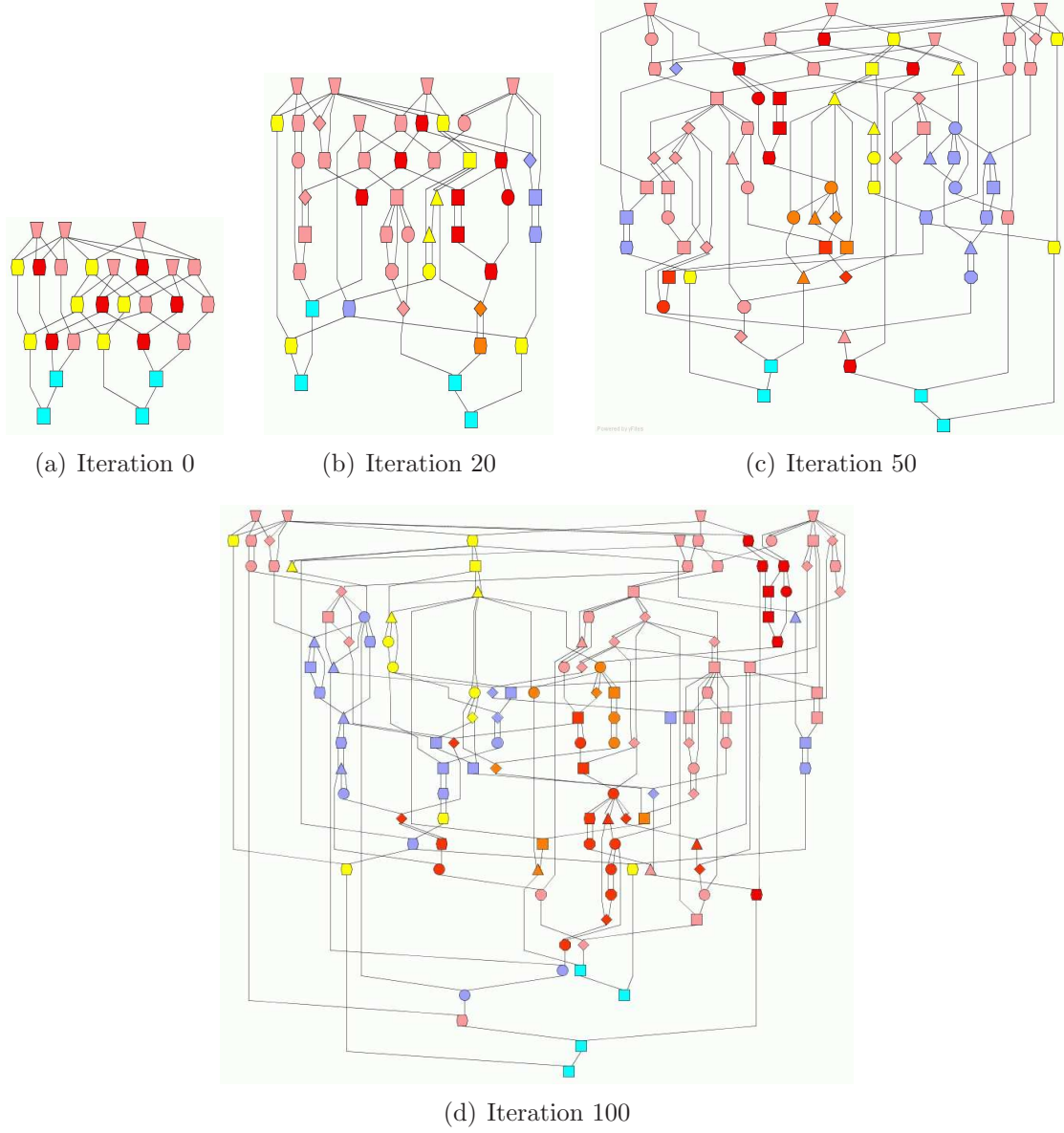
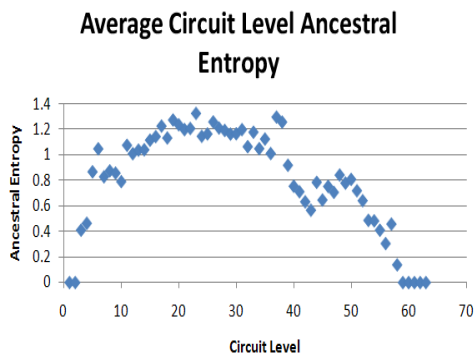
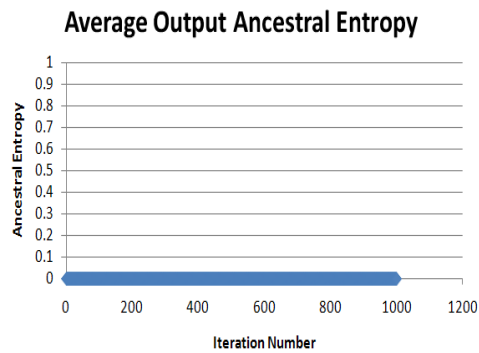


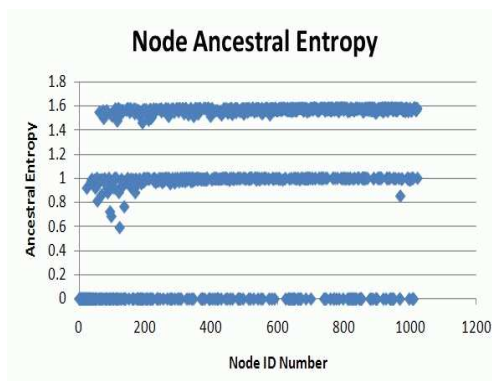
Figure B.9: LargestLevelTwoGates Shared Input C-17 Parallel Variant Circuits (a),(b),(c) and (d)



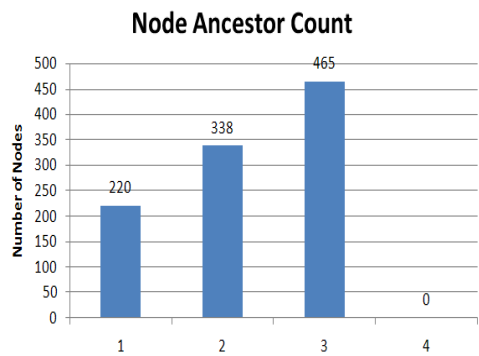
(a)



(b)

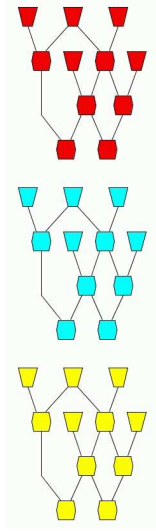


(c)

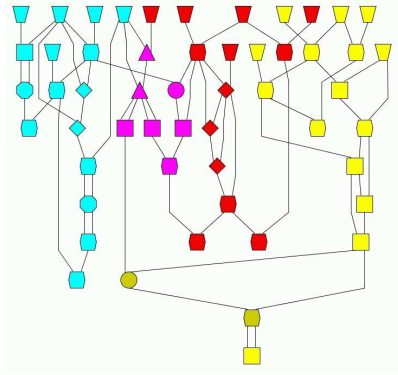


(d)

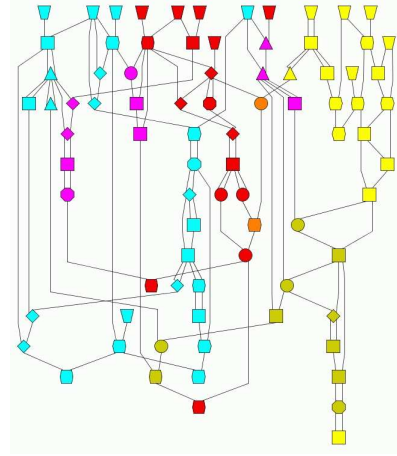
Figure B.10: LargestLevelTwoGates Shared Input C-17 Parallel - Iteration 1000
(a),(b),(c) and (d)



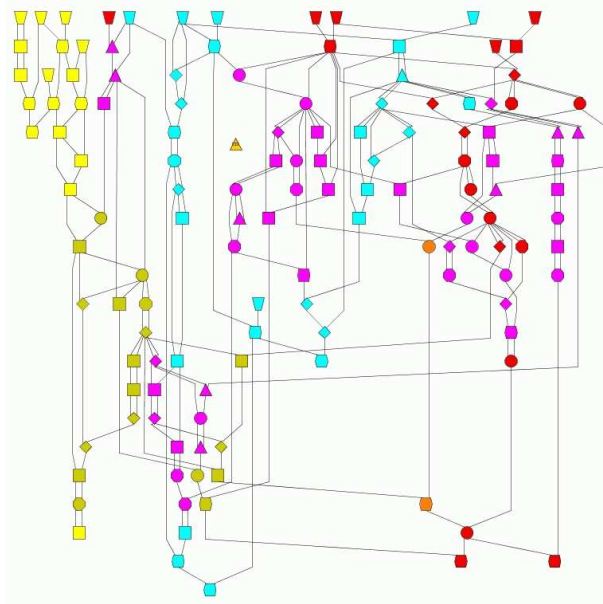
(a) Iteration 0



(b) Iteration 20



(c) Iteration 50



(d) Iteration 100

Figure B.11: LargestLevelTwoGates Individual Input C-17 Parallel Variant Circuits
(a),(b),(c) and (d)

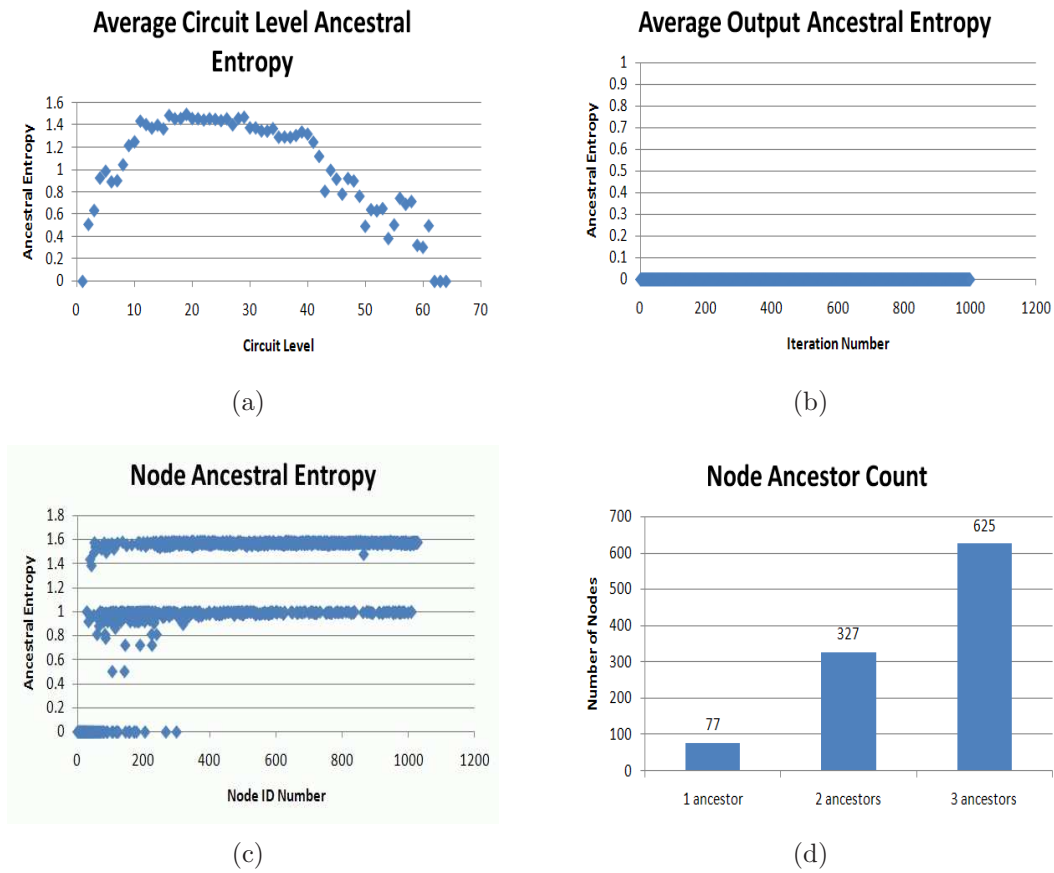


Figure B.12: LargestLevelTwoGates Individual Input C-17 Parallel - Iteration 1000
(a),(b),(c) and (d)

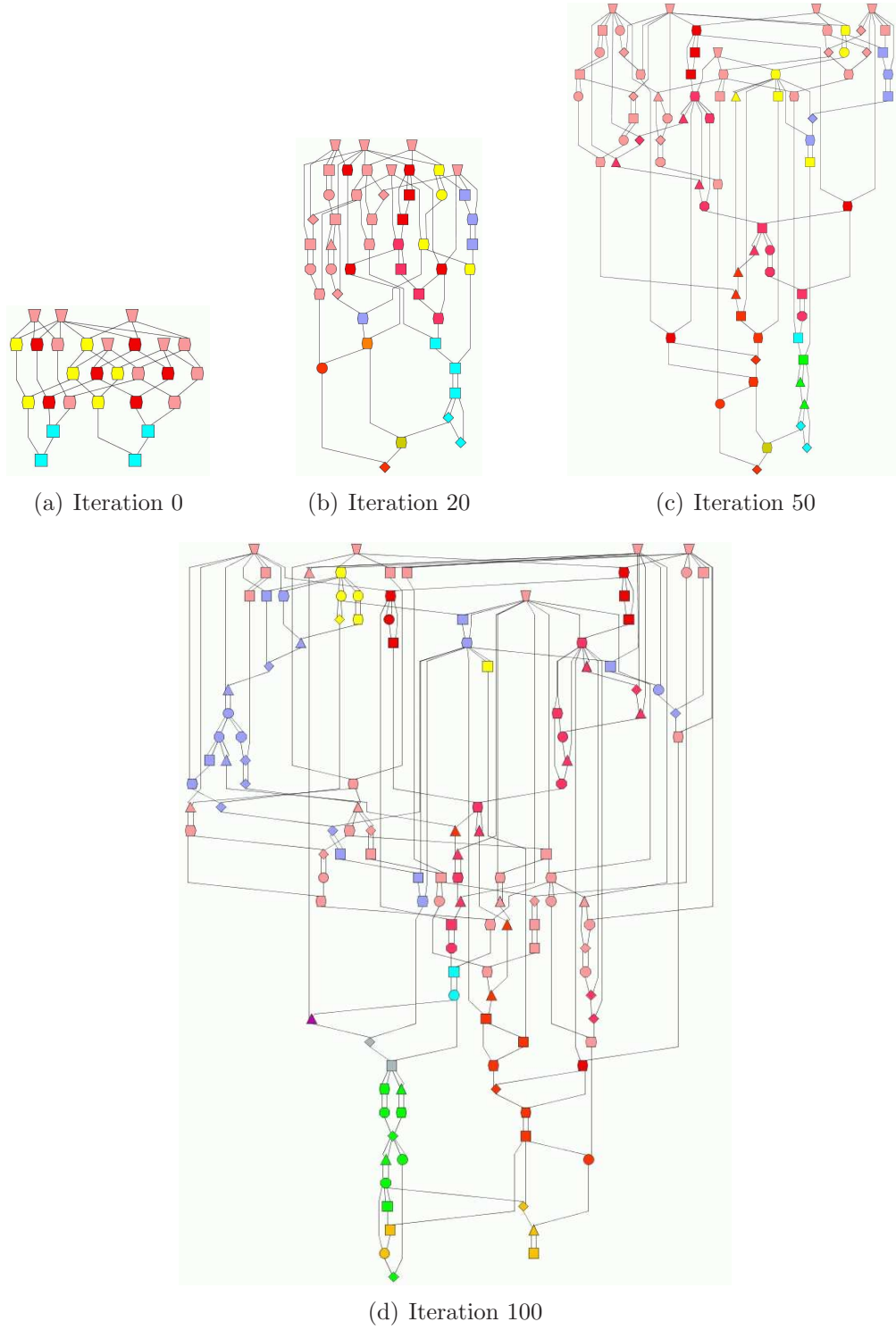
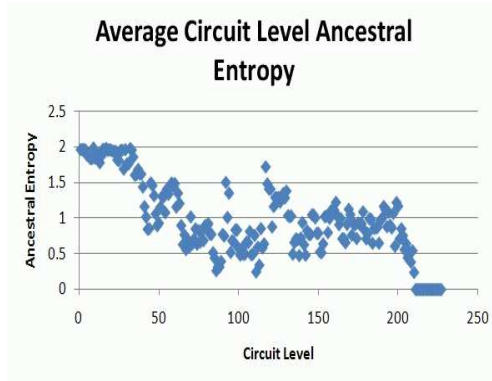
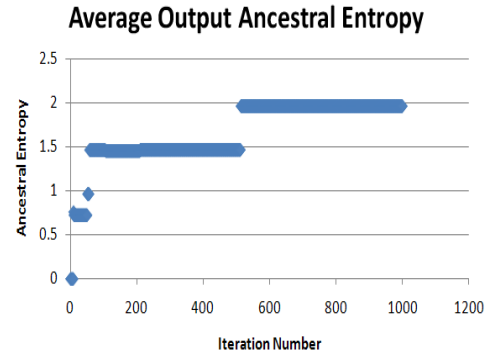


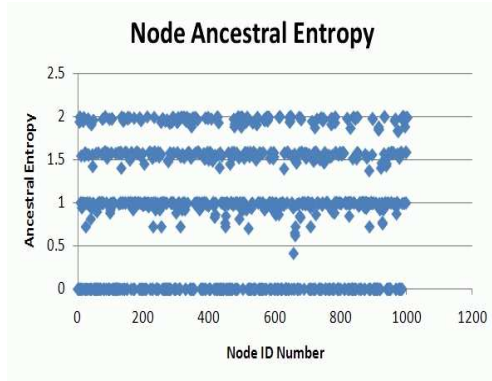
Figure B.13: RandomLevelTwoGates Shared Input C-17 Parallel Variant Circuits
(a),(b),(c) and (d)



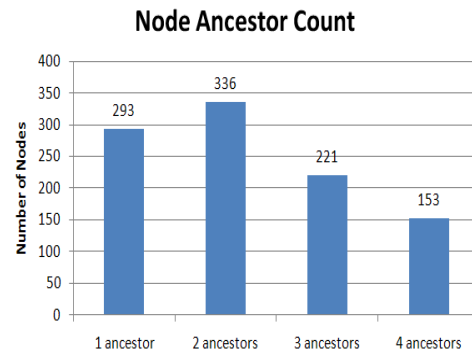
(a)



(b)

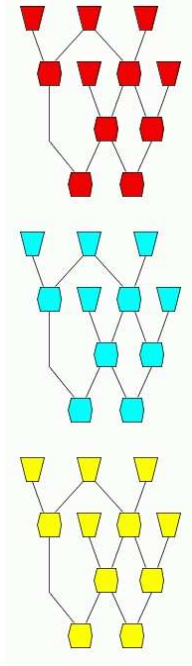


(c)

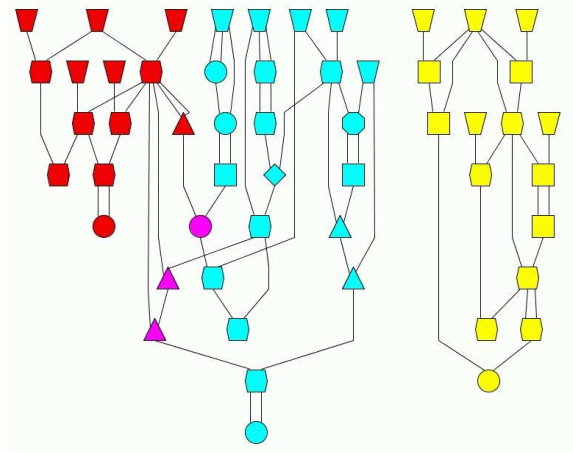


(d)

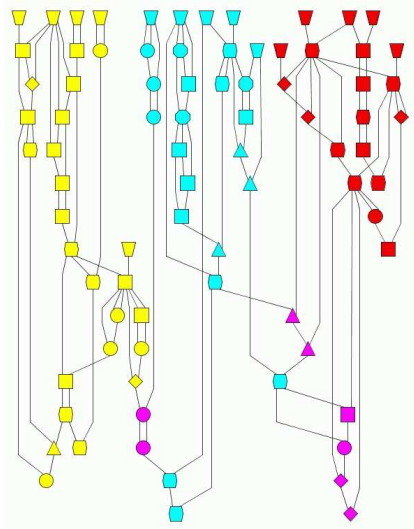
Figure B.14: RandomLevelTwoGates Shared Input C-17 Parallel - Iteration 1000
(a),(b),(c) and (d)



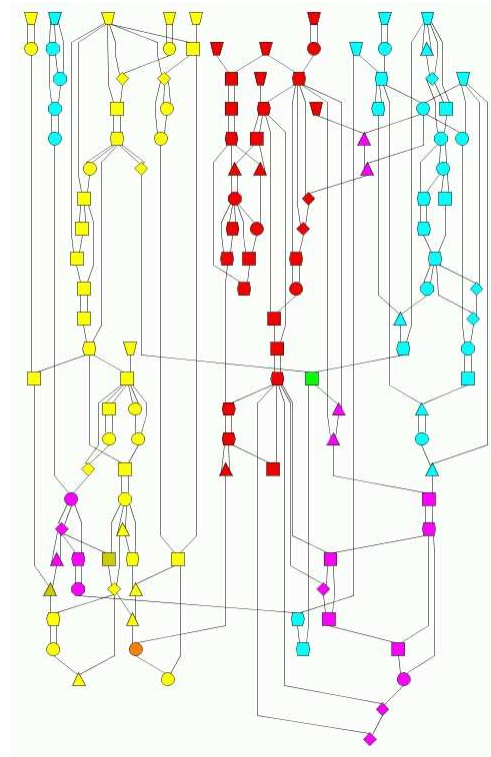
(a) Iteration 0



(b) Iteration 20

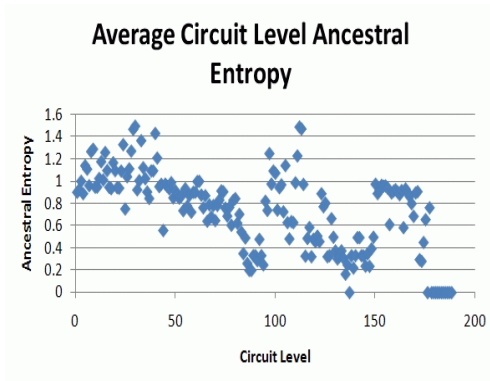


(c) Iteration 50

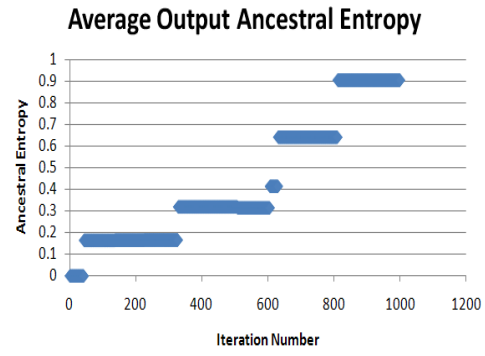


(d) Iteration 100

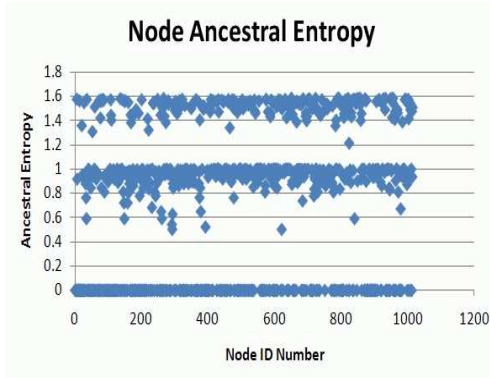
Figure B.15: RandomLevelTwoGates Individual Input C-17 Parallel Variant Circuits
(a),(b),(c) and (d)



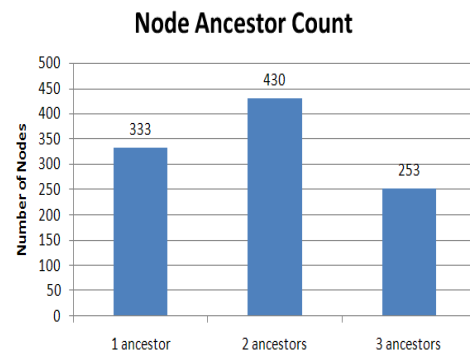
(a)



(b)



(c)



(d)

Figure B.16: RandomLevelTwoGates Individual Input C-17 Parallel - Iteration 1000
(a),(b),(c) and (d)

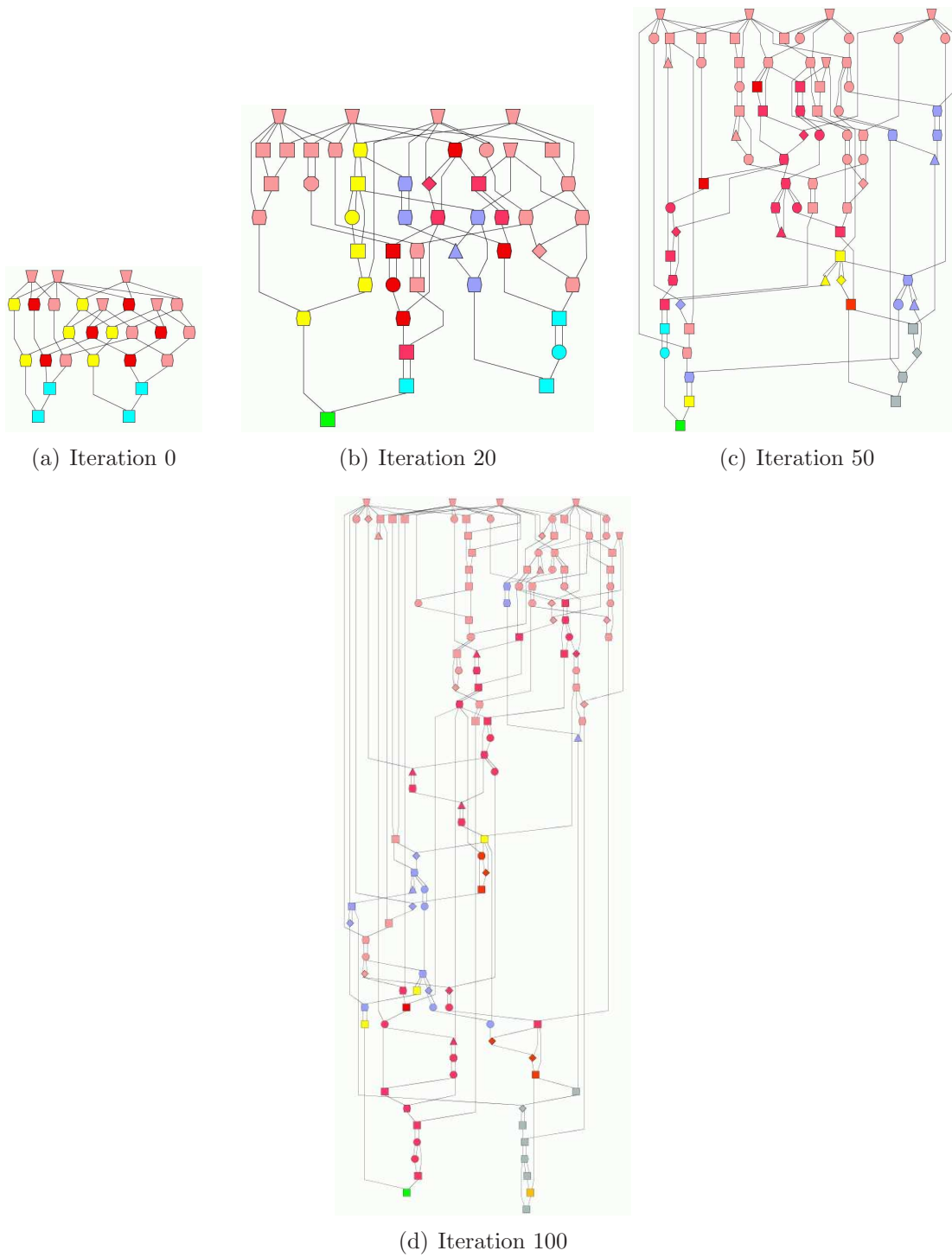
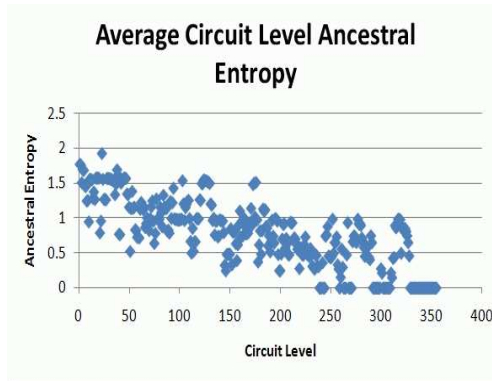
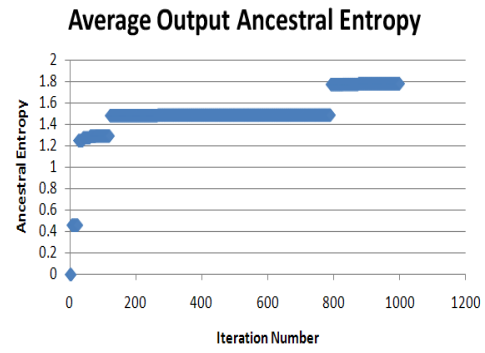


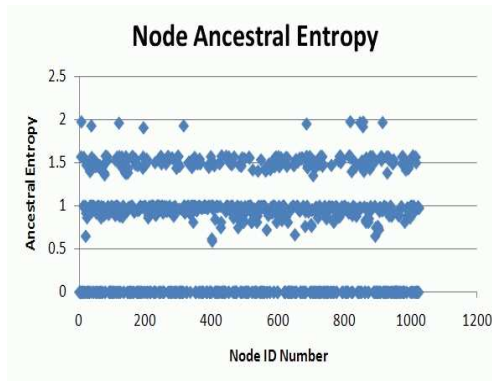
Figure B.17: RandomTwoGates Shared Input C-17 Parallel Variant Circuits
(a),(b),(c) and (d)



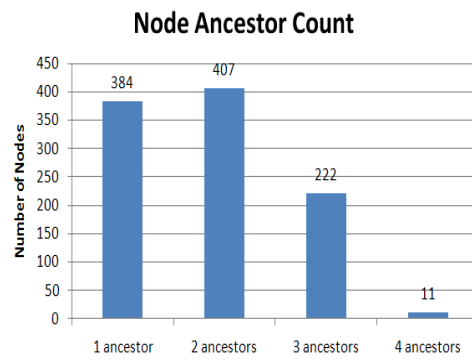
(a)



(b)

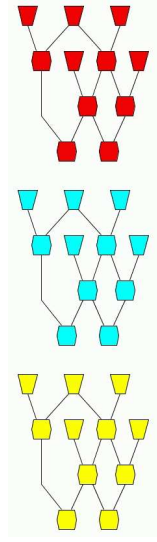


(c)

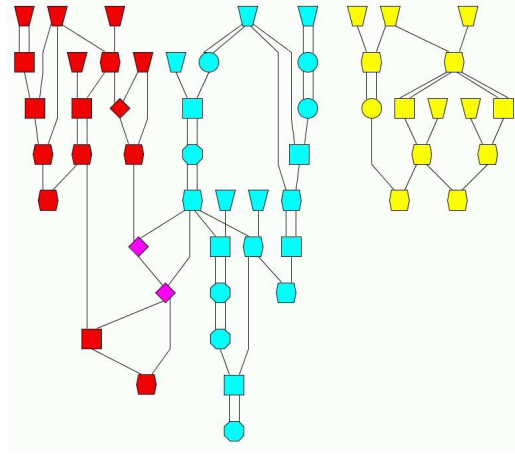


(d)

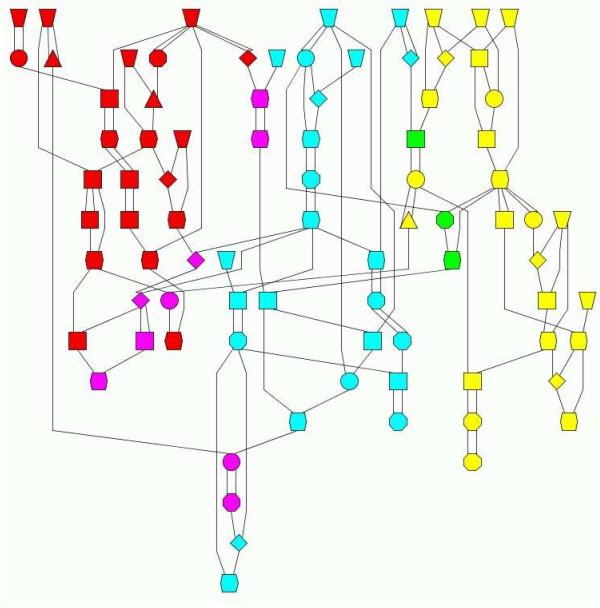
Figure B.18: RandomTwoGates Shared Input C-17 Parallel - Iteration 1000
(a),(b),(c) and (d)



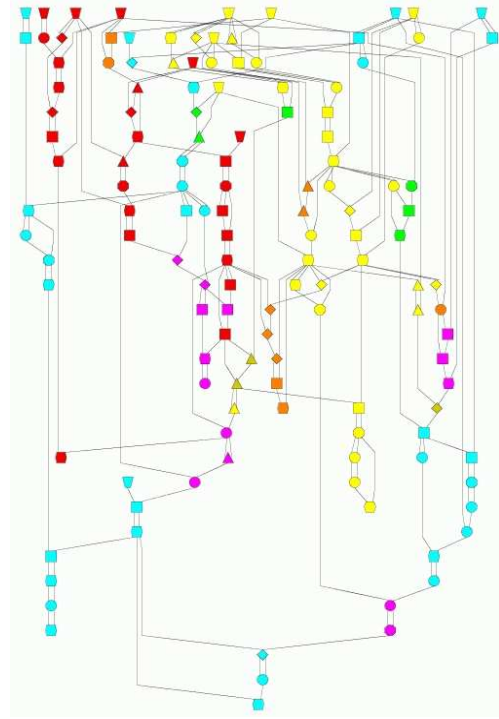
(a) Iteration 0



(b) Iteration 20

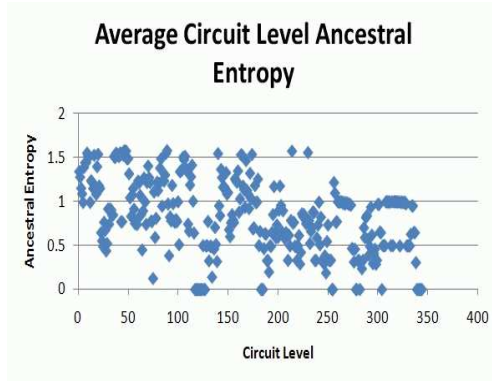


(c) Iteration 50

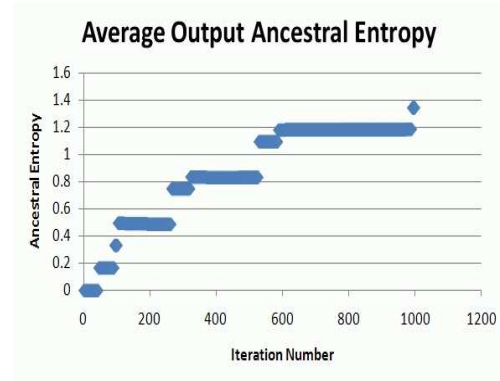


(d) Iteration 100

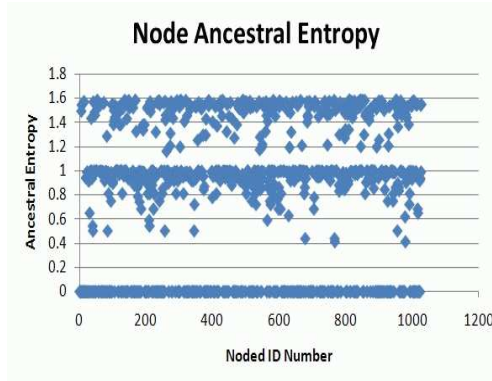
Figure B.19: RandomTwoGates Individual Input C-17 Parallel Variant Circuits
(a),(b),(c) and (d)



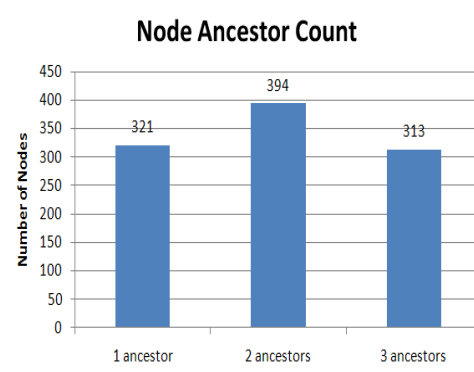
(a)



(b)



(c)



(d)

Figure B.20: RandomTwoGates Individual Input C-17 Parallel - Iteration 1000
(a),(b),(c) and (d)

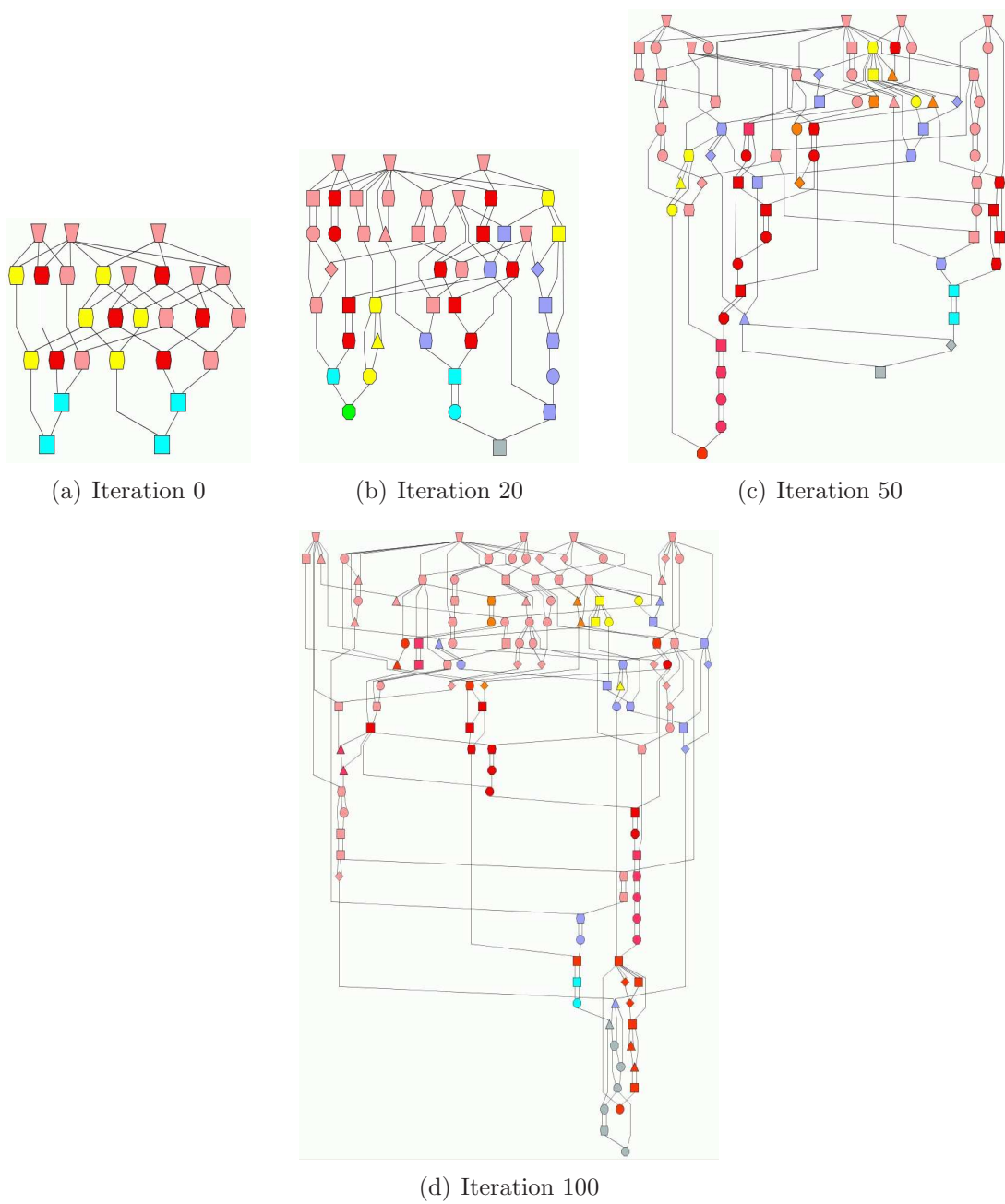
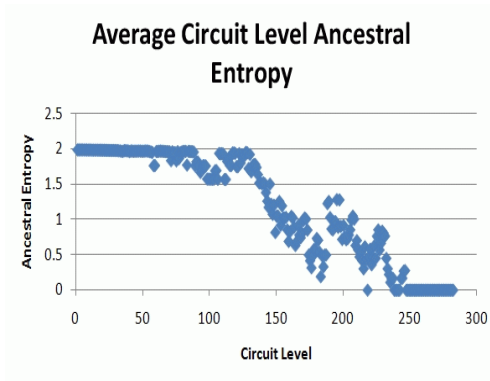
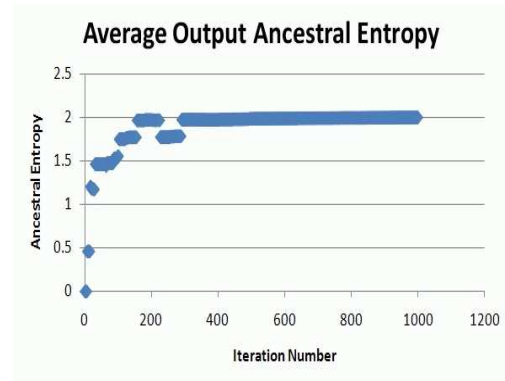


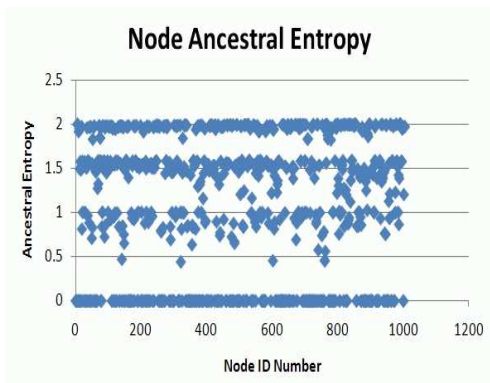
Figure B.21: RandomAlgorithm Shared Input C-17 Parallel Variant Circuits
(a),(b),(c) and (d)



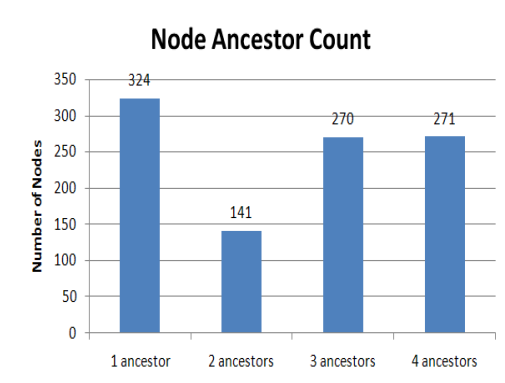
(a)



(b)

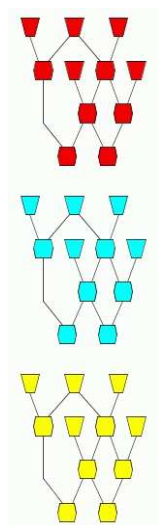


(c)

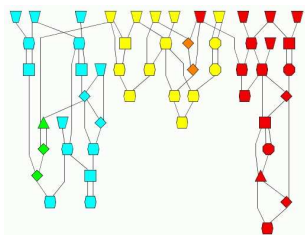


(d)

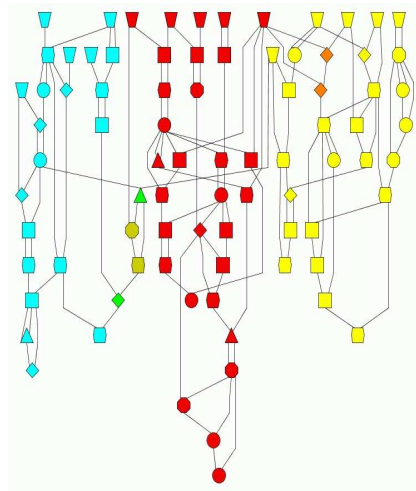
Figure B.22: RandomAlgorithm Shared Input C-17 Parallel - Iteration 1000
(a),(b),(c) and (d)



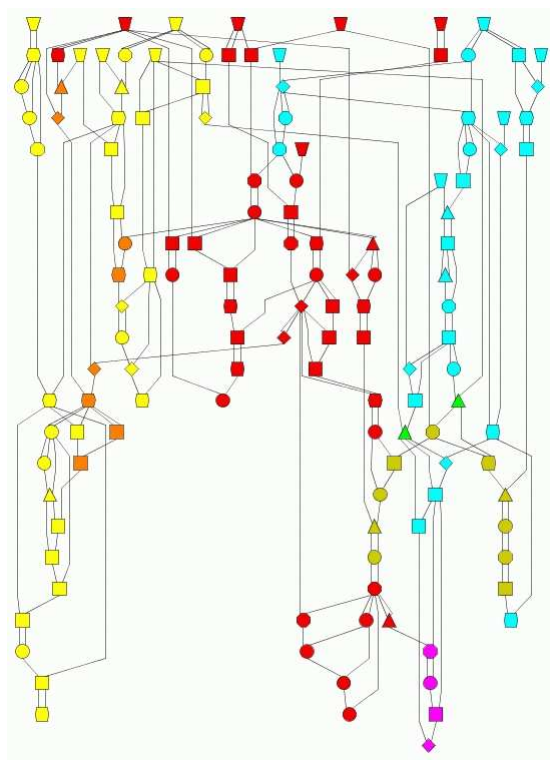
(a) Iteration 0



(b) Iteration 20

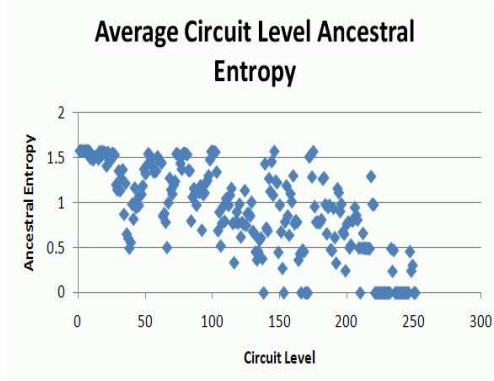


(c) Iteration 50

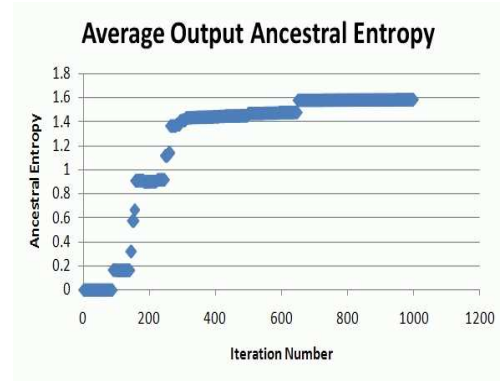


(d) Iteration 100

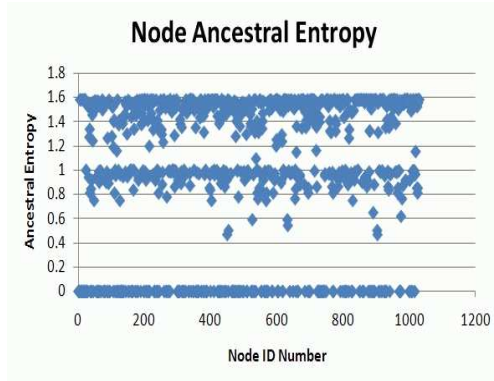
Figure B.23: RandomAlgorithm Individual Input C-17 Parallel Variant Circuits
(a),(b),(c) and (d)



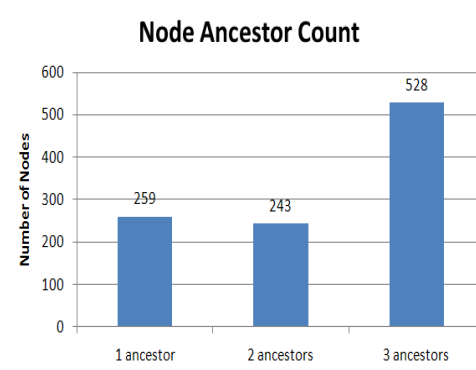
(a)



(b)



(c)



(d)

Figure B.24: RandomAlgorithm Individual Input C-17 Parallel - Iteration 1000
(a),(b),(c) and (d)

Appendix C. Four Gate Replacement Series and Parallel Circuit

Variant Charts and Graphs

This appendix contains the graphs and charts for the four gate replacement series and parallel circuits discussed in Chapter IV. These charts and graphs are from one experiment only and are provided to show trends of each algorithm.

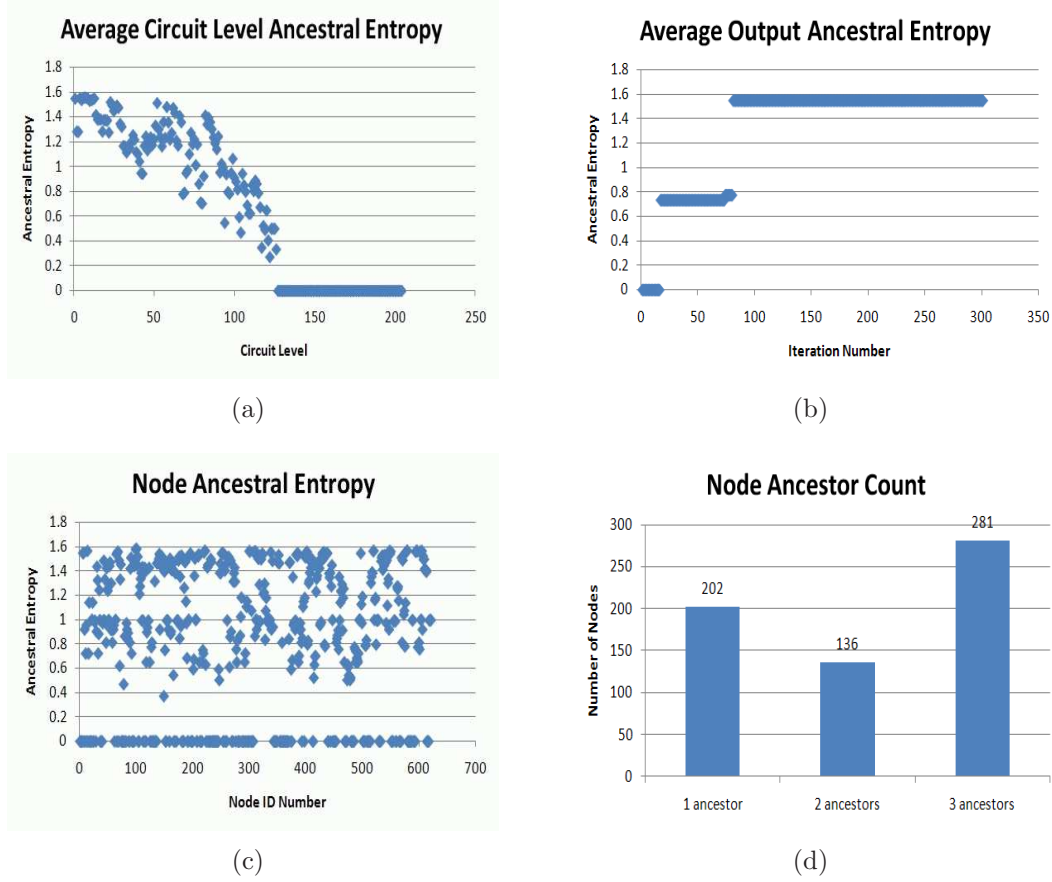
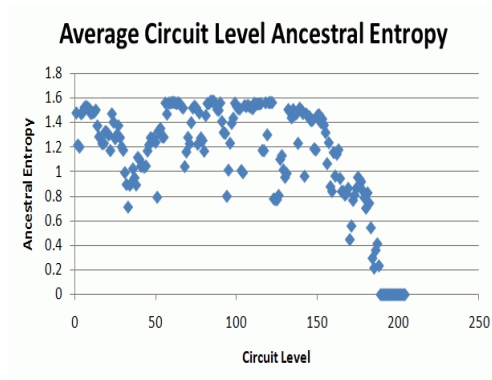
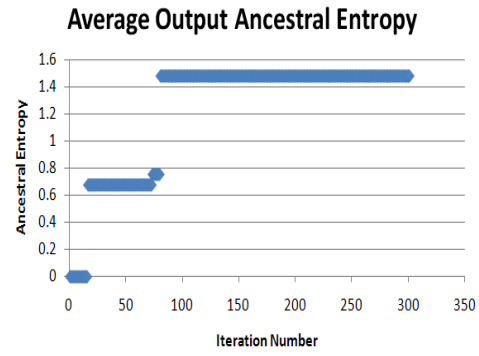


Figure C.1: RandomLevelTwoGates 5 Input C-17 Series 4 Gate Replacement - Iteration 1000

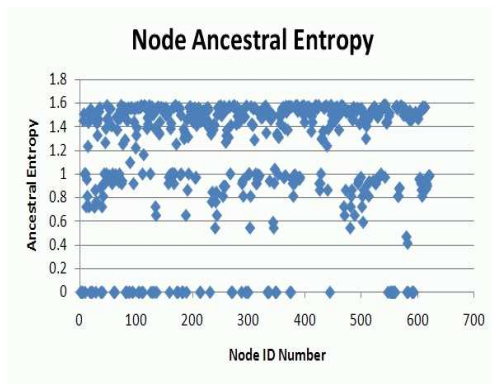
(a),(b),(c) and (d)



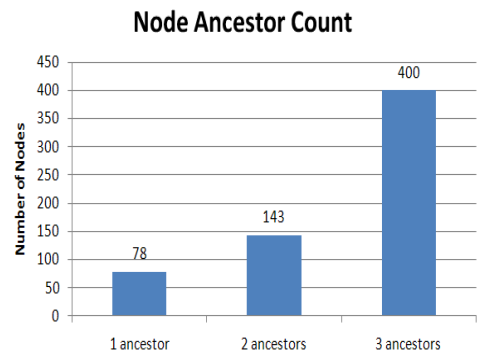
(a)



(b)



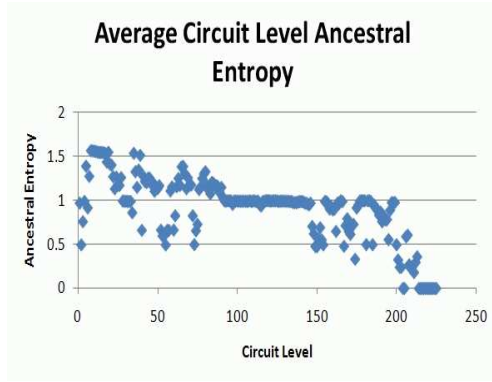
(c)



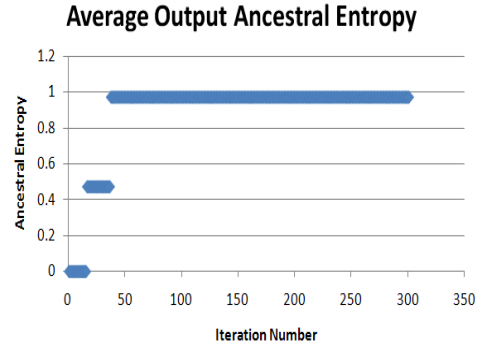
(d)

Figure C.2: RandomLevelTwoGates 5 ‘Split’ Input C-17 Series 4 Gate Replacement
- Iteration 1000

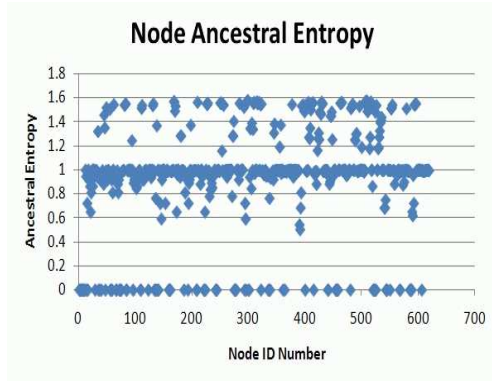
(a),(b),(c) and (d)



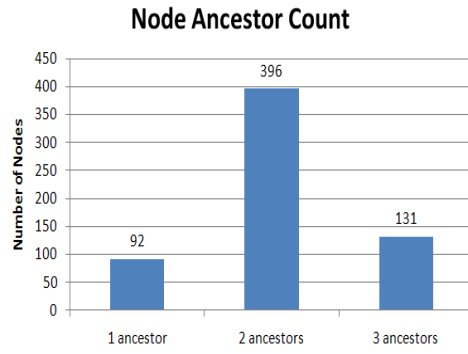
(a)



(b)



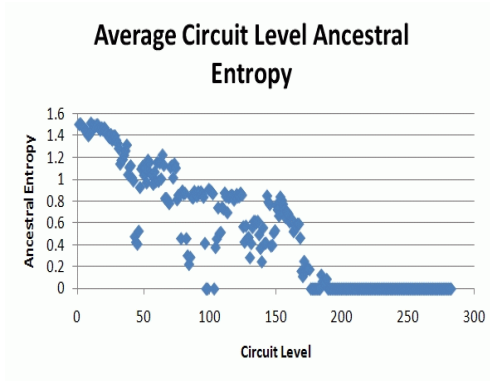
(c)



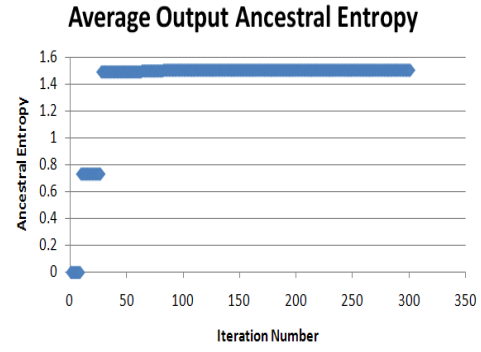
(d)

Figure C.3: RandomLevelTwoGates 11 Input C-17 Series 4 Gate Replacement - Iteration 1000

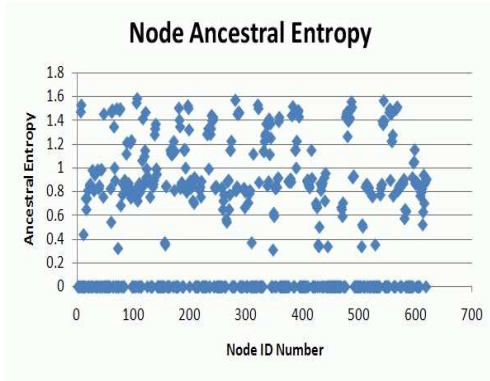
(a),(b),(c) and (d)



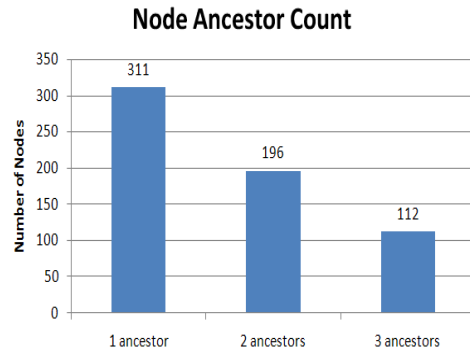
(a)



(b)



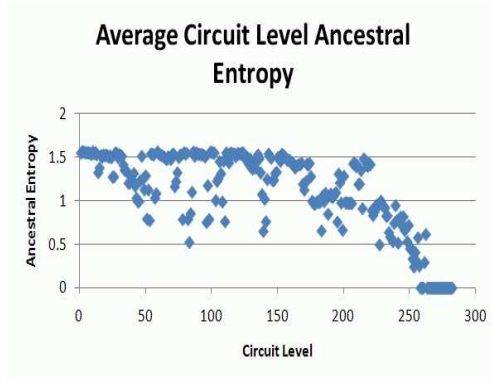
(c)



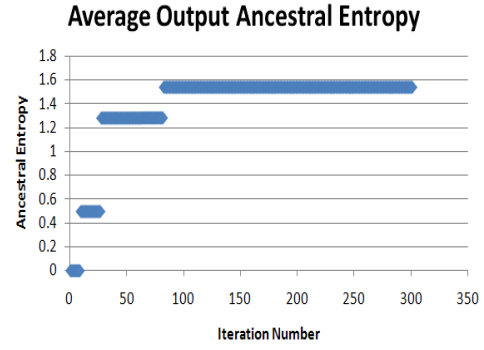
(d)

Figure C.4: RandomTwoGates 5 Input C-17 Series 4 Gate Replacement - Iteration 1000

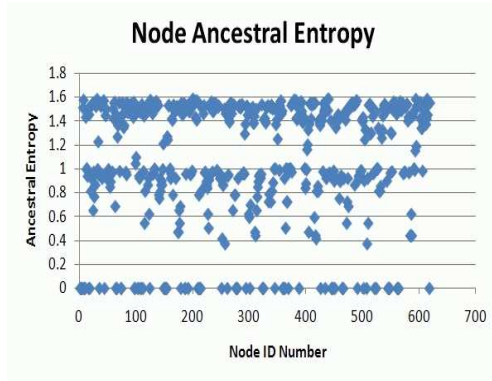
(a),(b),(c) and (d)



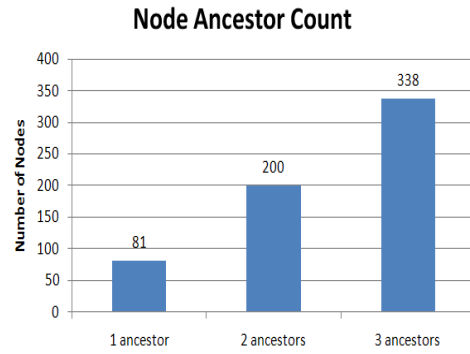
(a)



(b)



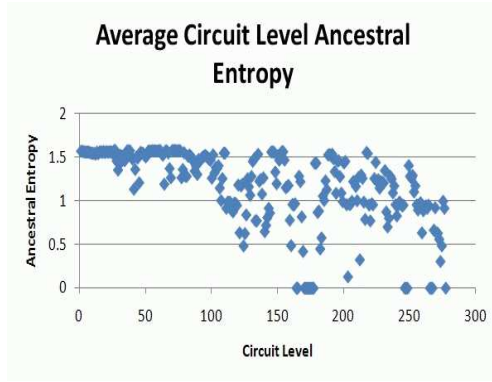
(c)



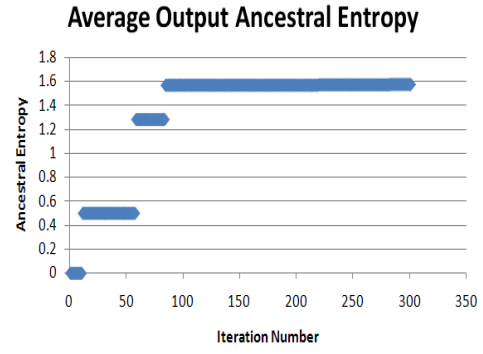
(d)

Figure C.5: RandomTwoGates 5 ‘Split’ Input C-17 Series 4 Gate Replacement - Iteration 1000

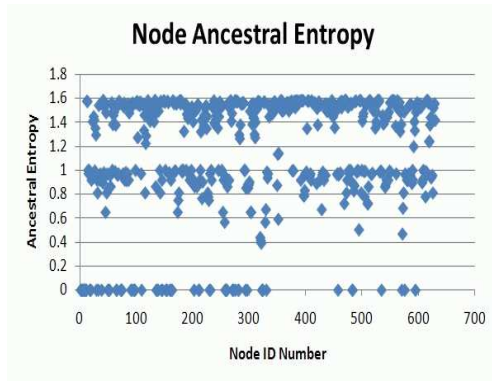
(a),(b),(c) and (d)



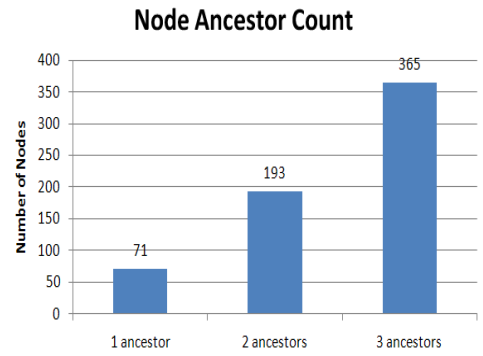
(a)



(b)



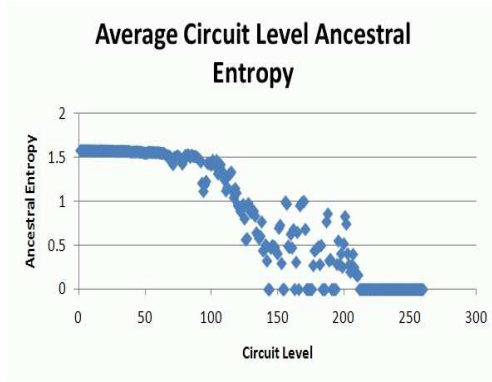
(c)



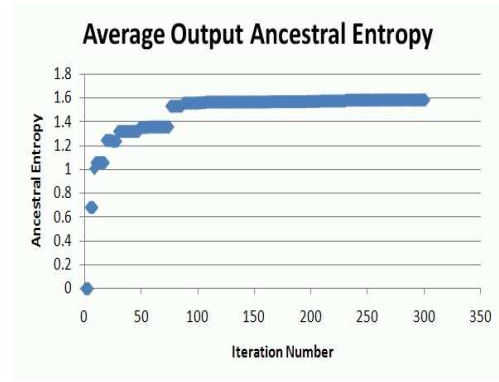
(d)

Figure C.6: Random Two Gate 11 Input C-17 Series 4 Gate Replacement - Iteration 1000

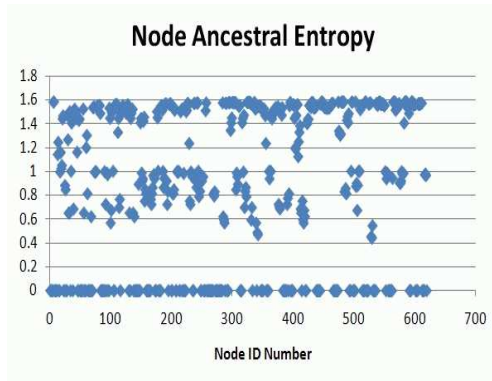
(a),(b),(c) and (d)



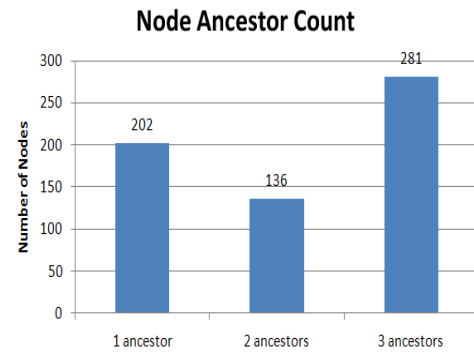
(a)



(b)



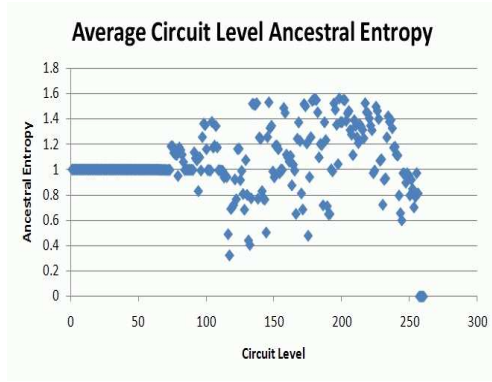
(c)



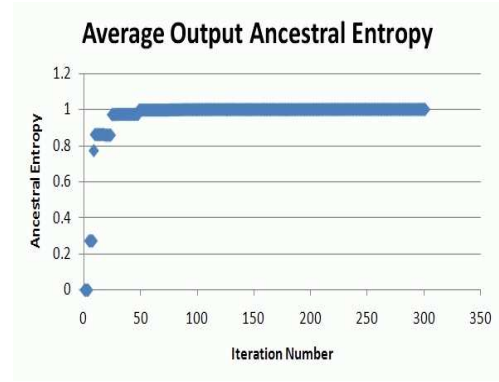
(d)

Figure C.7: RandomAlgorithm 5 Input C-17 Series 4 Gate Replacement - Iteration 1000

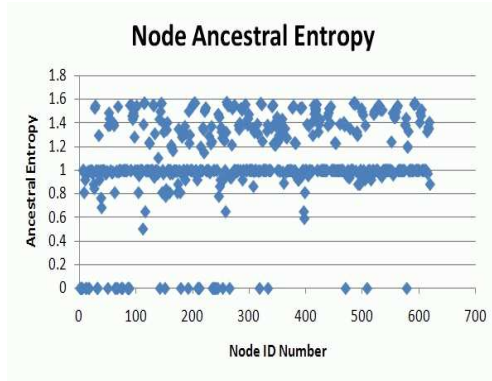
(a),(b),(c) and (d)



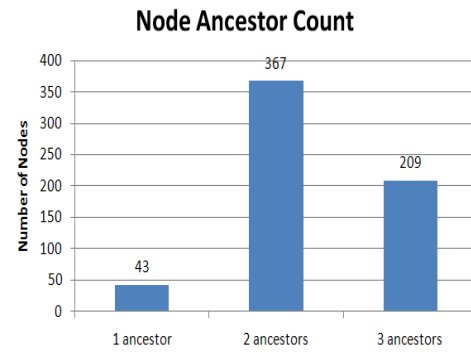
(a)



(b)



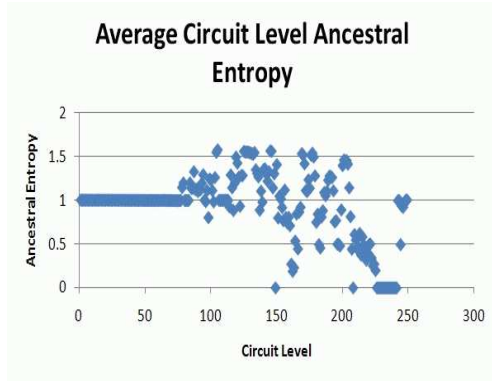
(c)



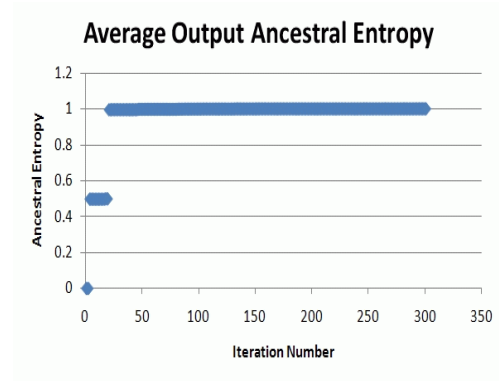
(d)

Figure C.8: RandomAlgorithm 5 ‘Split’ Input C-17 Series 4 Gate Replacement - Iteration 1000

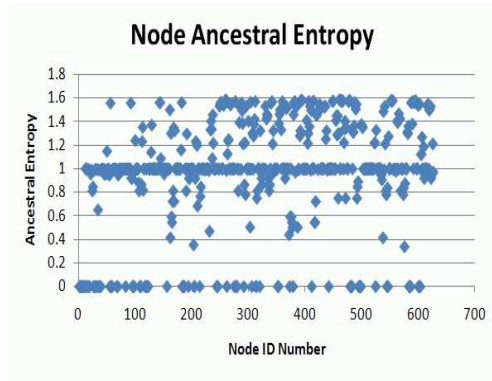
(a),(b),(c) and (d)



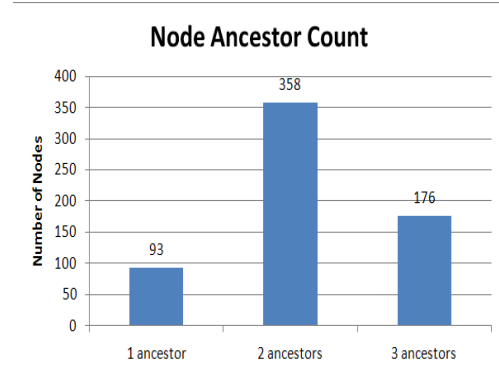
(a)



(b)



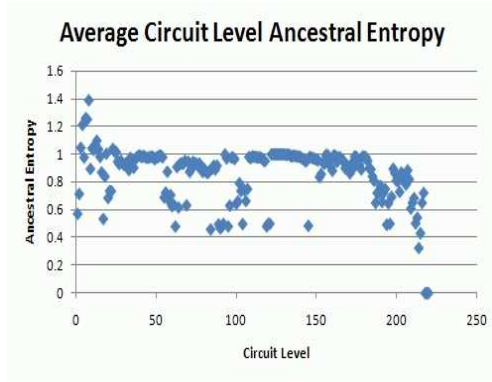
(c)



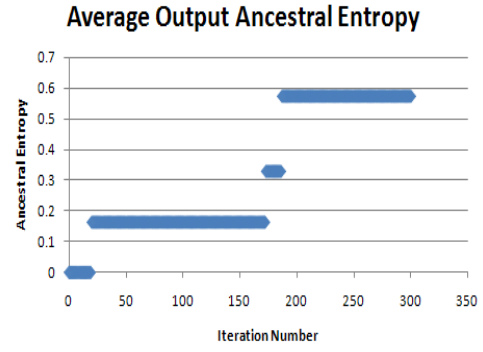
(d)

Figure C.9: RandomAlgorithm 11 Input C-17 Series 4 Gate Replacement - Iteration 1000

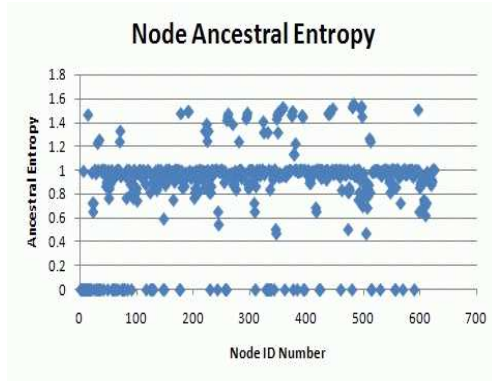
(a),(b),(c) and (d)



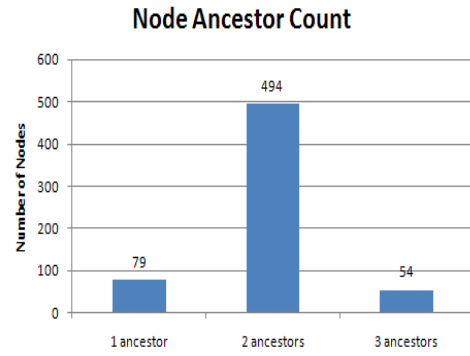
(a)



(b)



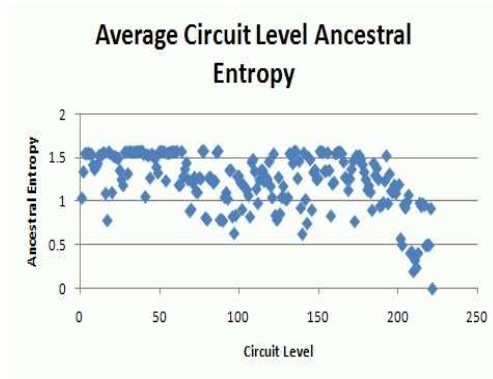
(c)



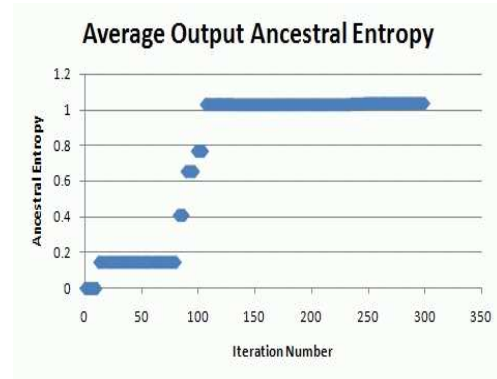
(d)

Figure C.10: 4 Gate Replacement RandomLevelTwoGates Individual Input C-17 Parallel - Iteration 1000

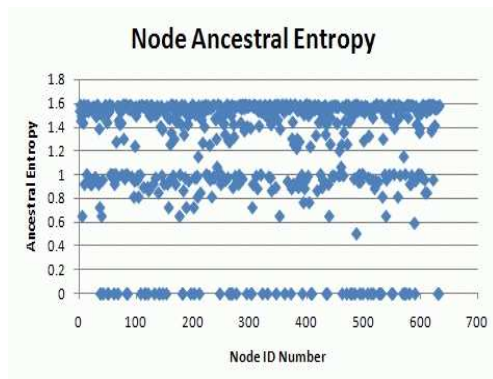
(a),(b),(c) and (d)



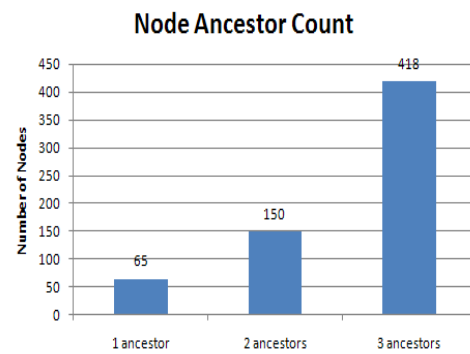
(a)



(b)

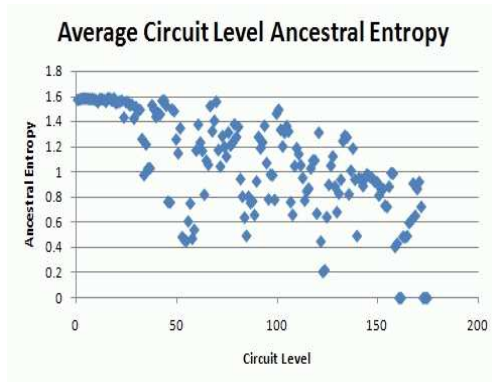


(c)

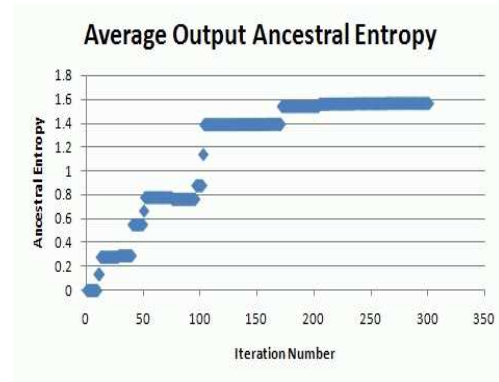


(d)

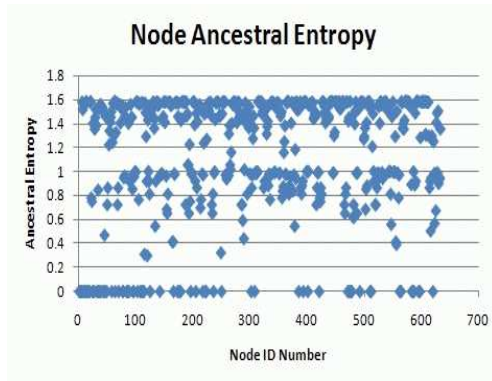
Figure C.11: 4 Gate RandomTwoGates Individual Input C-17 Parallel - Iteration 1000
(a),(b),(c) and (d)



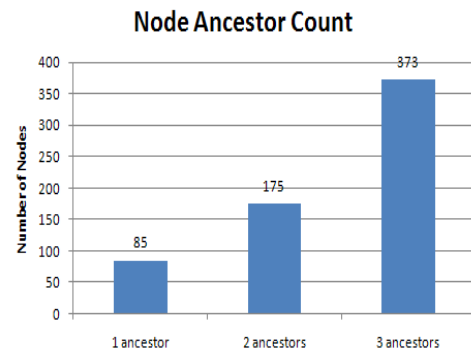
(a)



(b)



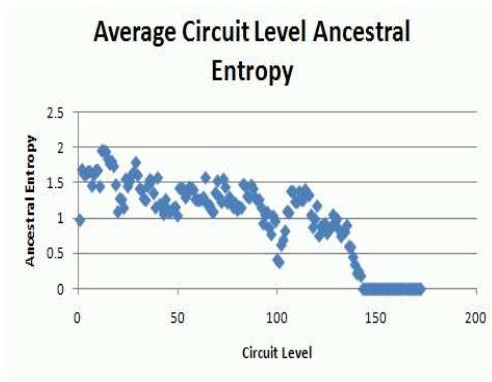
(c)



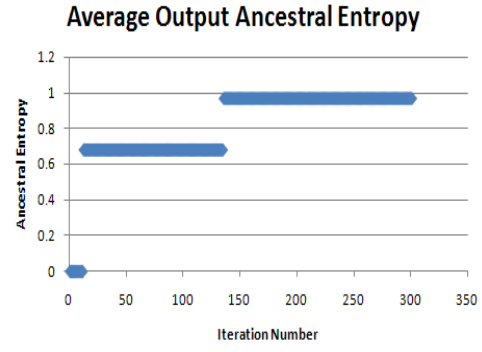
(d)

Figure C.12: 4 Gate RandomAlgorithm Individual Input C-17 Parallel - Iteration 1000

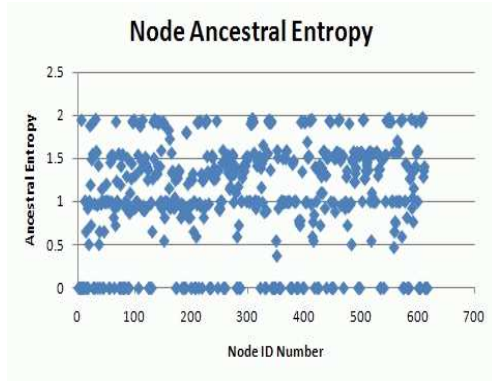
(a),(b),(c) and (d)



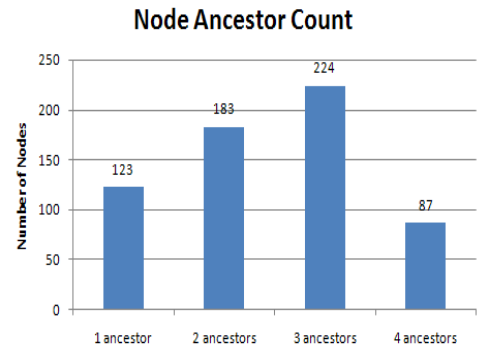
(a)



(b)



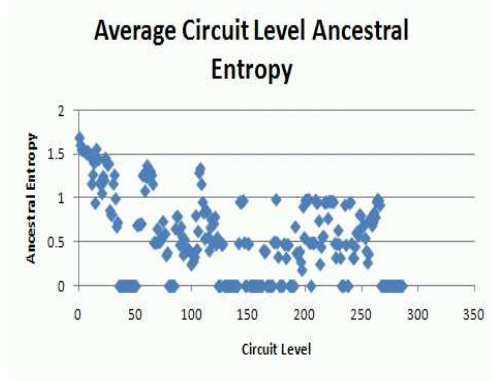
(c)



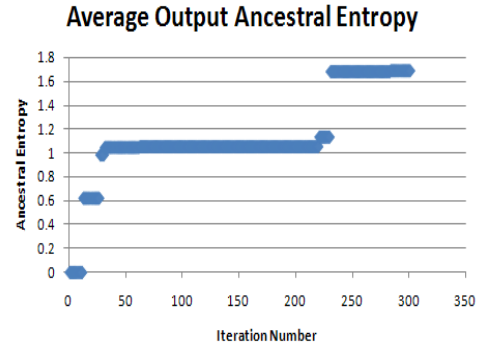
(d)

Figure C.13: 4 Gate RandomLevelTwoGates Shared Input C-17 Parallel - Iteration 1000

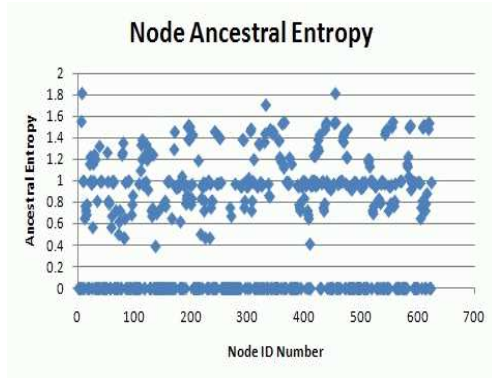
(a),(b),(c) and (d)



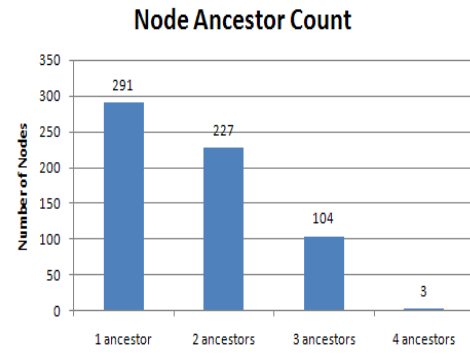
(a)



(b)

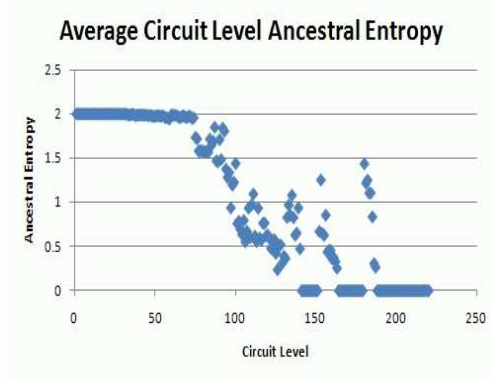


(c)

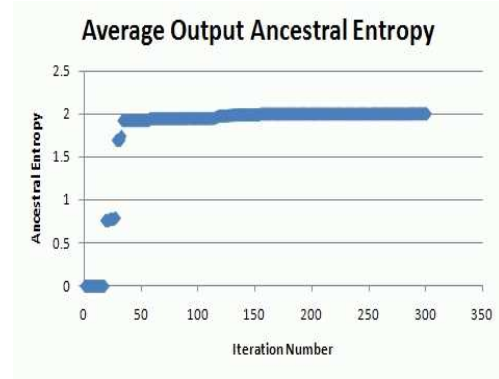


(d)

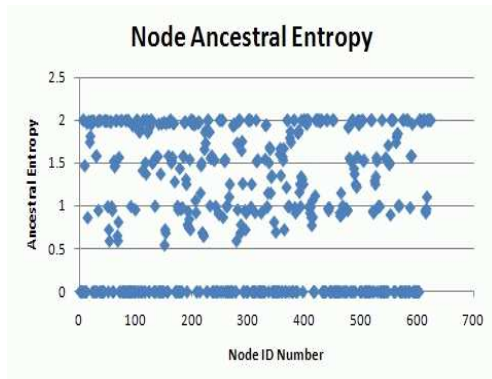
Figure C.14: 4 Gate RandomTwoGates Shared Input C-17 Parallel - Iteration 1000
(a),(b),(c) and (d)



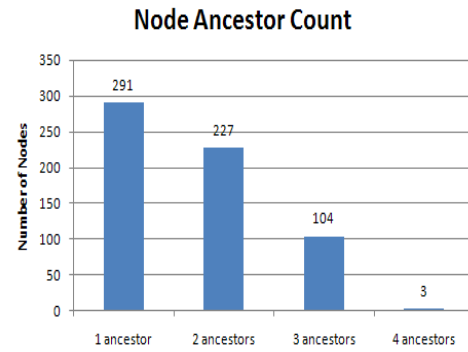
(a)



(b)



(c)



(d)

Figure C.15: 4 Gate RandomAlgorithm Shared Input C-17 Parallel - Iteration 1000
(a),(b),(c) and (d)

Bibliography

1. Agrawal, Dakshi, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. "The EM side-channel(s): Attacks and assessment methodologies", 2001. URL <http://www.research.ibm.com/intsec/emf-paper.ps>.
2. Agrawal, Dakshi, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. "The EM Side-Channel(s)". *CHES '02: Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, 29–45. Springer-Verlag, London, UK, 2003. ISBN 3-540-00409-2.
3. Chikofsky, Elliot J. and James H. Cross II. "Reverse Engineering and Design Recovery: A Taxonomy", January 1990.
4. Office of the Under Secretary of Defense, (Comptroller) Office. "RDT&E Programs (R-1)", 2007. URL <http://www.defenselink.mil/comptroller/defbudget/fy2008.r1.pdf>.
5. Directive, 5200.1-M DoD. *Acquisition Systems Program Protection*. the Assistant Secretary of Defense for Command, Control, Communications and Intelligence, March 1994.
6. Drimer, Saar. "Volatile FPGA design security – a survey (v0.96)", April 2008. URL http://www.cl.cam.ac.uk/~sd410/papers/fpga_security.pdf.
7. Gandolfi, K., C. Mourtel, and F. Olivier. "Electromagnetic Analysis: Concrete results", May 2001. URL <http://www.gemplus.com/smart/rd/publications/pdf/GM001ema.pdf>.
8. Hansen, Mark C., Hakan Yalcin, and John P. Hayes. "Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering". *IEEE Design and Test of Computers*, 16(3):72–80, 1999. URL <http://dx.doi.org/10.1109/54.785838>. Compilation and indexing terms, Copyright 2008 Elsevier Inc.
9. Kim, Han Seok. *Removing Redundant logic Pathways in Polymorphic Circuits*. Master's thesis, Air Force Institute of Technology, 2009.
10. Kukis, Mark and Katherine Arms. "Bush to China: Return Plane, Crew", April 2001. URL <http://www.military.com/Content/MoreContent1?file=standoff>.
11. Mayo, Wayland et al. "Russian B-29 Clone - The TU-4 Story", 2008. URL <http://www.rb-29.net/HTML/03RelatedStories/03.03shortstories/>.
12. McDonald, Jeffrey T. *Enhanced Security for Mobile Agent Systems*. Ph.D. thesis, Florida State University, 2006.
13. Mish, Frederick C. (editor). *Merriam-Webster's collegiate dictionary*. Merriam-Webster, Incorporated, Springfield, MA, 10 edition, 2001. ISBN 0-87779-710-2.

14. Nohl, Karsten, David Evans, and Henryk Plotz. "Reverse-Engineering a Cryptographic RFID Tag". 187–193. 2008.
15. Norman, Ken. *Architecture for White-box Obfuscation Using Randomized Subcircuit Selection And Replacement*. Master's thesis, Air Force Institute of Technology, 2008.
16. "PreEmptive Solutions", Jan 2008. URL <http://www.preemptive.com/>.
17. Rajgopal, Suresh and Akhilesh Tyagi. "Spatial Entropy - A Unified Attribute to Model Dynamic Communication in VLSI Circuits (Under the direction of Kye S. Hedlund and Akhilesh Tyagi)", 1992.
18. Rauch, John G. "THE LAW ON REVERSE ENGINEERING". *IEEE Spectrum*, 30(8):47–48, -08 1993.
19. Rekoff, M. G. Jr. "ON REVERSE ENGINEERING." (*Univ of Alabama in Birmingham, Dep of Electrical Engineering, Birmingham, AL, USA*) Source: *IEEE Transactions on Systems, Man and Cybernetics*, SMC-15(2):244, -04 1985. Doi: pmid:.
20. Scott, Jeff. "Buran Space Shuttle", November 2003. URL <http://www.aerospaceweb.org/question/spacecraft/q0153.shtml>.

Vita

Captain Jason A. Williams graduated from Ashland-Greenwood High School in Ashland, Nebraska. He entered undergraduate studies at the University of Nebraska in Lincoln, Nebraska where he graduated with a Bachelor of Science degree in Computer Engineering in 2003. He was commissioned through Officer Training School in 2004.

Captain Williams entered the US Air Force in 1990. He was first assigned to the 558th Civil Engineering Squadron, Nellis AFB, Nevada as a Metal Fabrication Specialist. He was assigned as an instructor to the 366th Training Squadron, Detachment 6, at the Naval Construction Training Center, Mississippi in September 1995. He was awarded the Naval Construction Training Center Instructor of the year and the 366th Training Squadron Junior Instructor of the year in 1997. He also earned the Master Instructor Certificate in 1998. His next assignment was at Kunsan AB, South Korea, where he was assigned as a Facilities Maintenance Manager in 2000. While in Korea, he was selected to participate in the Airman Education Commissioning Program. His next assignment was at the University of Nebraska, Air Force ROTC Detachment 465, Nebraska, in 2001. He completed a Bachelor of Science Degree in Computer Engineering in 2003. After completing Officer Training School in 2004, he was assigned to the Air Armament Center, Eglin AFB, Florida. In 2006 he was re-assigned to the Air Force Research Laboratory at Eglin AFB. While there, he was selected to attend the Air Force Institute of Technology at Wright-Patterson AFB, Ohio and was assigned there in 2007. Upon graduation, he will remain at Wright-Patterson AFB for his assignment to the Air Force Research Laboratory.

Permanent address: 2950 Hobson Way
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433

| REPORT DOCUMENTATION PAGE | | | | | <i>Form Approved</i> OMB No. 0704-0188 | |
|---|--------------------|--|-----------------------------------|---|---|--|
| The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS. | | | | | | |
| 1. REPORT DATE (DD-MM-YYYY) 26-03-2009 | | 2. REPORT TYPE Master's Thesis | | 3. DATES COVERED (From — To) May 2007–Mar 2009 | | |
| 4. TITLE AND SUBTITLE Characterizing Component Hiding Using Ancestral Entropy | | | | 5a. CONTRACT NUMBER | | |
| | | | | 5b. GRANT NUMBER | | |
| | | | | 5c. PROGRAM ELEMENT NUMBER | | |
| 6. AUTHOR(S) Williams, Jason A. | | | | 5d. PROJECT NUMBER 08-183 | | |
| | | | | 5e. TASK NUMBER | | |
| | | | | 5f. WORK UNIT NUMBER | | |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765 | | | | 8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCE/ENG/09-12 | | |
| 9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Dr. Robert L. Herklotz Air Force Office of Scientific Research, AFMC 801 North Randolph Street, Rm 732 Arlington VA 22203-1977 703-696-9544 (DSN: 426) robert.herklotz@afosr.af.mil | | | | 10. SPONSOR/MONITOR'S ACRONYM(S) AFOSR/NL | | |
| | | | | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) | | |
| 12. DISTRIBUTION / AVAILABILITY STATEMENT Approval for public release; distribution is unlimited. | | | | | | |
| 13. SUPPLEMENTARY NOTES | | | | | | |
| 14. ABSTRACT <p style="font-size: x-small;"> In this research, the problem of software protection and the attributes that define that protection is considered. Specifically, how to protect programs defined as structural combinational logic gates. Obfuscation is one technique for protecting such circuits and involves replacing an original circuit with a functionally equivalent variant that has some definable hiding property. The difficulty of reverse engineering versus identifying and recovering the original components or sub-circuits within an original circuit is compared. With a polymorphic circuit engine that produces semantically equivalent variations of standard benchmark circuits the level of component hiding across variants with different physical configurations is determined to provide an entropy-based attribute to assess whether components are merged at the structural level. Specific types of obfuscating transformations with respect to component hiding using ancestral entropy are compared as well as the measure of uncertainty related to origination of a gate within a circuit. </p> | | | | | | |
| 15. SUBJECT TERMS software obfuscation, component hiding, entropy, ancestry, combinational circuit | | | | | | |
| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON | |
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | Lt Col J. Todd McDonald | |
| U | U | U | UU | 201 | 19b. TELEPHONE NUMBER (include area code) 937-255-3636 x4639, jmcdonal@afit.edu | |